ESCUELA SUPERIOR DE INGENIERÍA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

SDLgolf:
Videojuego de golf en 2D

Andrés Francisco Aparicio

28 de abril de 2011

ESCUELA SUPERIOR DE INGENIERÍA

INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS

SDLgolf:
Videojuego de golf en 2D

- Departamento: Lenguajes y sistemas informáticos
- Director del proyecto: José Luis Isla Montes
- Autor del proyecto: Andrés Francisco Aparicio

Cádiz, 28 de abril de 2011

Fdo: Andrés Francisco Aparicio

Agradecimientos

A mis padres, Andrés y María Jesús, por todo el esfuerzo que han realizado para permitirme llegar hasta aquí.

A toda mi familia, especialmente a mis hermanos, por ayudarme desde siempre en todo lo que han podido.

A D. José Luis Isla Montes, por las facilidades que me ha brindado desde el primer momento, así como su ayuda, atención e interés en dirigir este proyecto.

Licencia

Índice general

Índice de figuras

Capítulo 1

Introducción

En la actualidad, el mundo del software libre está viviendo la expansión más importante de toda su historia.

La aparición de las distribuciones Live CD[1] a mediados de los años 90, permitieron un primer acercamiento al sistema operativo GNU/Linux por parte de los usuarios principiantes.

En un primer momento, dicho sistema operativo estaba reservado a los gurús de la informática. Hoy en día, es fácil encontrar a cualquier persona que, consciente o inconscientemente, use habitualmente GNU/Linux.

Teléfonos móviles[2], videoconsolas[3], ordenadores portátiles[4]... cada vez son más los dispositivos que funcionan sobre éste sistema operativo libre, lo que a su vez supone un incremento constante en el número de personas que lo usan a diario.

1.1. Motivación

Mi primer contacto con el software libre fue en los años en los que cursaba la Educación Secundaria Obligatoria. En aquel entonces, llevaba varios años usando ordenadores que funcionaban con sistemas operativos propietarios como MS-DOS o la serie Microsoft Windows y tenía algunos conocimientos de programación. Tuve la fortuna de

[1] Yggdrasil Linux, la primera distribución Live CD Linux, aunque Knoppix supuso la verdadera revolución http://www.knopper.net/knoppix-info/index-en.html

[2] Android, el sistema operativo basado en Linux, es la plataforma más usada en teléfonos inteligentes y su crecimiento en el mercado es cada vez mayor http://www.canalys.com/pr/2011/r2011013.html

[3] Consolas portátiles como Pandora http://openpandora.org/ o la saga GP producidas por la empresa surcoreana GamePark Holdings http://www.openhandhelds.org/index.php están basadas en el sistema operativo libre Linux

[4] Actualmente, la mayoría de fabricantes incorporan en sus ordenadores portátiles sistemas Linux embebidos de arranque inmediato que permiten el uso del software más frecuente como editores de texto, visores de imágenes, navegadores, VoIP, mensajería instantánea, etc. http://en.wikipedia.org/wiki/Instant_on

que llegara a mis manos un ordenador "*COMPAQ Presario 2200*"[5] e instalar una "*Mandrake Linux*" que me permitió iniciar una flamante terminal. Desde aquel momento, mi interés por el software libre no dejó de ir en aumento.

Ya en la Universidad de Cádiz, desde un primer momento, se nos inculcó la importancia que tenía el software libre, así como las ventajas que todo ello aportaba.

Mientras realizaba mis estudios de Ingeniero Técnico en Informática de Sistemas, descubrí la asignatura optativa "*Diseño de Videojuegos*" de **D. Manuel Palomo Duarte**. Este hecho, unido a mi interés por el mundo de los videojuegos (desde que tuve acceso a una *Nintendo NES*), propiciaron enormemente el aprendizaje del desarrollo de un videojuego así como la adquisición de las habilidades, técnicas y conocimientos necesarios para la creación del mismo.

Todo lo anteriormente expuesto fue lo que me llevó a la idea de desarrollar un juego libre empleando como estandarte el conjunto de bibliotecas SDL y el lenguaje de programación C++.

1.2. Propósito

El objetivo principal del proyecto es el desarrollo de un videojuego de código fuente abierto usando herramientas y recursos exclusivamente libres.

Dado que actualmente no existe ningún juego de golf en 2D para sistemas Linux, decidí llevar a cabo SDLgolf, un juego de género deportivo, subgénero arcade[6], de tipo multijugador, diseño bidimensional y destinado a jugadores casuales[7].

Como ya se indicó en la sección anterior, el juego ha sido desarrollado en el lenguaje de programación orientado a objetos C++, empleándose principalmente la biblioteca SDL[8] junto a las librerías auxiliares de sonido (SDL_mixer), imagen (SDL_image) y fuentes (SDL_ttf), y TinyXML[9] para la lectura y escritura de datos.

[5] Procesador Cyrix 180Mhz, 16MiB de RAM, cualquier teléfono móvil inteligente supera estas especificaciones técnicas...

[6] Los videojuegos arcade son juegos relativamente fáciles de jugar y que no se corresponden fielmente con la realidad. Es el tipo de juegos que predominaba en las máquinas recreativas (también conocidas como máquinas arcade)

[7] Los jugadores casuales no tienen conocimientos sobre términos relacionados con los videojuegos y buscan un juego simple para jugar partidas de corta duración con el fin de entretenerse

[8] SDL es un conjunto de bibliotecas desarrolladas en el lenguaje de programación C que proporcionan capas de abstracción para operaciones de dibujo bidimensionales, efectos de sonido, música e imágenes.

[9] Analizador sintáctico de documentos XML. Pequeño, potente y de código abierto.

SDLgolf ha sido liberado bajo licencia GPL v3 (GNU General Public License). Se incluyen los términos de la licencia en inglés al final del documento[10].

1.3. Acerca de este documento

Este documento se organiza en los siguientes capítulos:

- En el capítulo 1, se comentan las razones que han motivado la creación de este proyecto, así como el propósito del mismo.
- En el capítulo 2, se definen conceptos básicos relacionados con los videojuegos y se hace una breve reseña histórica sobre los videojuegos libres.
- En el capítulo 3, se comenta la planificación del proyecto y su organización temporal.
- En el capítulo 4, se determinan las funciones básicas del juego y los requisitos del sistema software.
- En el capítulo 5, se describe el análisis del sistema y los casos de uso.
- En el capítulo 6, se define el diseño del sistema.
- En el capítulo 7, se exponen las dificultades encontradas a lo largo de la implementación del videojuego y cómo se han solventado.
- En el capítulo 8, se detallan las pruebas a las que se ha sometido el sistema.
- En el capítulo 9, se muestra la valoración personal del proyecto, las mejoras que tendrán lugar en el futuro y las herramientas que se han empleado en el desarrollo.
- Se incluye un Manual de usuario en el que se detallan los pasos para instalar el videojuego y las instrucciones para usarlo.
- Se incluye un apéndice en el que se explica el Formato de los documentos XML que se emplean en el videojuego.

[10] Para más información visite http://www.gnu.org/licenses

Capítulo 2

Conceptos básicos

2.1. Definición de videojuego

Un videojuego es un tipo de software destinado, principalmente, al entretenimiento de una o varias personas. Los videojuegos se ejecutan en dispositivos electrónicos como videoconsolas, ordenadores o teléfonos móviles y permiten a los jugadores la interacción con el sistema.

2.1.1. Géneros

La clasificación de los videojuegos en diferentes géneros puede realizarse atendiendo a muy diversos factores, como pueden ser el sistema de juego, los objetivos que han de llevarse a cabo, el tipo de interacción entre el jugador y la máquina, el diseño de los gráficos, etc.

El constante avance en la tecnología de los videojuegos y la aparición de nuevos títulos surgidos de la fusión de diferentes géneros, hace que cada vez sea más difícil la clasificación de los mismos.

A continuación, haremos una descripción de los géneros más conocidos:

- **Acción**: En los juegos de acción es necesario tener buenos reflejos, puntería y saber administrar el tiempo. Este género es el más básico y uno de los más extendidos.

  De este tipo de juego derivan multitud de subgéneros como el Beat 'em up, en el que el jugador se va enfrentando cuerpo a cuerpo a diferentes grupos de enemigos, los juegos de lucha, donde un luchador pelea contra otro, los de disparo, en los que se debe avanzar disparando a los enemigos, los juegos de laberintos, los de pinball o los plataformas, en los que el jugador tiene que ir recogiendo objetos y moverse por una serie de plataformas para superar los niveles.

Figura 2.1: SuperTux:
Videojuego de plataformas inspirado en Super Mario Bros

- **Aventura**: En este tipo de juego, el jugador asume el rol del protagonista de una aventura interactiva, en la que tendrá que ir avanzando mediante la exploración del entorno, la resolución de puzzles y la interacción con otros personajes del juego.

  En este tipo de juegos, se definen otros subgéneros como las aventuras conversacionales, las aventuras gráficas y las aventuras de rompecabezas.

Figura 2.2: Advent:
Un motor de aventuras gráficas libre, similar a Monkey Island

- **Acción-Aventura**: Los juegos de acción-aventura combinan los elementos del género acción con los del género aventura, es decir, en este tipo de juegos se desafiarán tanto los reflejos como la capacidad de resolver problemas.

Dentro de este género se diferencian los Survival Horror, en los que se recrea una atmósfera de terror psicológico, los plataformas de aventura, los plataformas de vista isométrica y los juegos de rol-acción.

Figura 2.3: Ardentryst:
Videojuego de rol-acción de código abierto

- **Deportes**: Los videojuegos de deportes simulan los juegos tradicionales de deporte. Es uno de los géneros más populares que existen y también uno de los más vendidos en el mercado. Todos los deportes conocidos han sido recreados en el mundo de los videojuegos.

Figura 2.4: Tuxfootball:
Juego de fútbol en 2D libre

- **Estrategia**: En los juegos de estrategia, el jugador hará uso de sus habilidades de planteamiento, razonamiento y lógica para controlar, desde una visión global de

la superficie de juego, a todas las unidades que estén a su mando a fin de poder derrotar al enemigo y alcanzar la victoria. El sistema de combate puede ser en tiempo real o por turnos y suele enfocarse en la estrategia militar o táctica.

Los juegos de artillería y construcción de imperios derivan directamente de este género.

Figura 2.5: Freeciv:
Videojuego de estrategia por turnos de código abierto

- **Rol**: En los juegos de rol, el jugador controla a uno o varios personajes y tendrá que realizar misiones de diversa índole, enfrentándose a los adversarios en combates cuerpo a cuerpo o mediante el empleo de armas y poderes mágicos. Mientras que se avanza en la historia, el personaje irá mejorando sus aptitudes físicas y místicas e irá especializándose en determinadas tareas.

  Este género abarca los videojuegos de rol multijugador masivos en línea, en los que los jugadores se adentran en un mundo virtual en el que pueden colaborar entre ellos para llevar a cabo las misiones, los juegos de rol tácticos, en los que se incorporan elementos de los videojuegos de estrategia, o los juegos de rol-acción, que fusionan las características de los juegos de rol clásicos con combates en tiempo real.

Figura 2.6: The Mana World:
Videojuego de rol multijugador masivo online libre

- **Simulación**: Este tipo de videojuego simula determinados aspectos de la realidad, como pueden ser actividades de la vida cotidiana u otros aspectos no tan frecuentes.

  Dentro de este género se incluyen los simuladores de vida, los de construcción y gestión, los juegos de simulación económica, juegos de simulación deportiva, simuladores de vuelo y de conducción y muchos tantos otros.

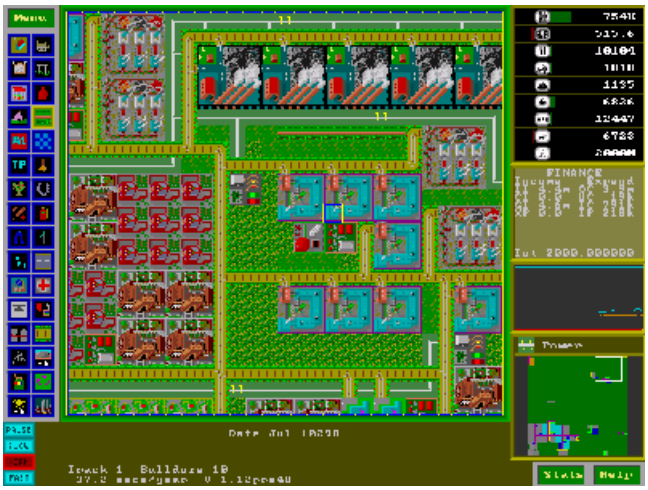


Figura 2.7: Lincity:
Constructor de ciudades de código abierto

2.2. Definición de videojugador

Un videojugador es una persona que juega a videojuegos.

2.2.1. Tipos

Los jugadores pueden clasificarse en los siguientes tipos:

- **Newbie**: Personas con poca o ninguna experiencia en un determinado juego o en el mundo de los videojuegos en general.
- **Casual**: Jugador cuyo tiempo o interés por los videojuegos es limitado. Los jugadores casuales tienden a jugar videojuegos sencillos y a realizar partidas de corta duración.
- **Gamer**: Es un tipo de jugador muy experimentado que dedica mucho tiempo a la práctica de videojuegos y se decanta por juegos complejos y de larga duración. Además de esto, los gamers están al tanto de todas las novedades relacionadas con los videojuegos, formando parte de esa cultura. También suelen participar en ligas o torneos.
- **Progamer**: Videojugador profesional que juega por dinero. Un jugador profesional es económicamente dependiente de los ingresos que consigue jugando, por lo que, a diferencia del resto de jugadores, el tiempo que dedica a jugar no es su tiempo libre.

2.3. Videojuegos de código abierto

Los videojuegos de código abierto son aquellos que poseen un código fuente abierto. La mayoría de ellos son de libre distribución y, normalmente, multiplataformas.

Se denominan juegos libres a aquellos juegos que, a parte de ser de código abierto, contienen recursos exclusivamente libres. Casi todos los juegos de código abierto son software libre, pero algunos de ellos no lo son, debido a que incluyen recursos propietarios.

2.3.1. Antecedentes

Por norma general, los videojuegos libres son desarrollados por pequeños grupos de personas en sus ratos libres y sin la intención de obtener beneficios económicos.

Algunos juegos libres están basados en juegos propietarios que fueron liberados[1], aunque la mayor parte de ellos provienen de proyectos voluntarios en los que los programadores son claros entusiastas de los videojuegos.

2.3.2. Breve historia

Como en la mayoría de los otros tipos de software, el concepto de videojuego libre surgió de manera inconsciente durante la creación de los primeros juegos, especialmente, los desarrollados para el sistema operativo Unix[2], siendo gran parte de ellos proyectos originales o clones de los juegos arcade y de las aventuras conversacionales de la época.

A mediados de los años 80, el auge del software propietario condujo al desarrollo de videojuegos totalmente libres como GNU Chess[3], que pertenecía al proyecto GNU, el cual tenía entre sus objetivos crear sistemas software completamente libres, incluyendo videojuegos. Otros fueron sumándose al movimiento GNU, tales como NetHack[4] o Netrek[5].

A principios de los años 90, empezaron a desarrollarse juegos más avanzados, en los que se hacía uso del sistema de ventanas X. Aparecieron títulos como XBill[6], uno de los primeros juegos activistas que se crearon o XEvil, surgido inicialmente de un proyecto universitario. Rocks'n'Diamonds[7] fue otro de los primeros juegos libres que se crearon para Linux.

---

[1] Juegos como Wolfenstein Enemy Territory, videojuego multijugador de disparos en primera persona http://en.wikipedia.org/wiki/Wolfenstein:_Enemy_Territory, o Warzone 2100, juego de estrategia en tiempo real http://wz2100.net/, pasaron a tener licencia GPL

[2] Un claro ejemplo de esto es la colección de juegos BSD Games, en la que se incluyen títulos de aventura, tablero, cartas, puzzle, rol, etc. http://wiki.linuxquestions.org/wiki/BSD_games

[3] Uno de los juegos de ajedrez más antiguos para sistemas Unix

[4] Videojuego de rol de mazmorras, con gráficos ASCII, en el que los niveles se generan aleatoriamente http://www.nethack.org/

[5] Juego 2D de estrategia en tiempo real, multijugador y con ambientación espacial http://www.netrek.org/

[6] Videojuego arcade en el que hay que impedir que un virus llamado Wingdows se expanda por los diferentes sistemas http://www.xbill.org/

[7] Juego de lógica en el que se controla a un personaje que tiene que cavar y recolectar una serie de diamantes para que la puerta de salida al siguiente nivel se abra http://www.artsoft.org/rocksndiamonds/

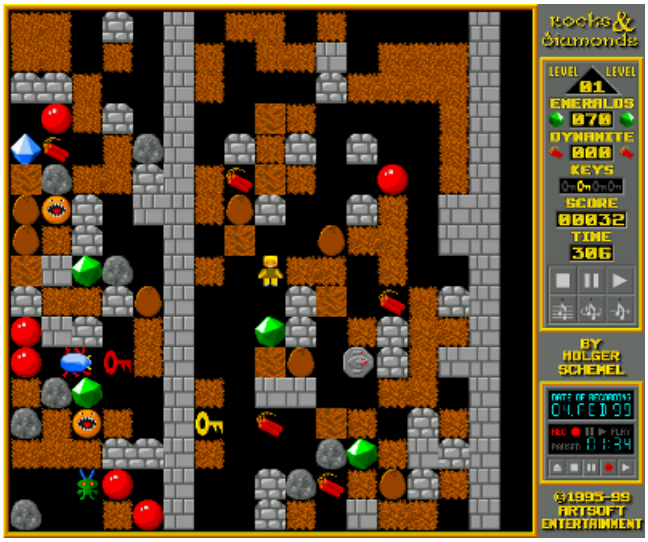


Figura 2.8: Rocks'n'Diamonds:
Uno de los primeros juegos para Linux

A mediados de los años 90 y con el salto a la era tridimensional, los primeros títulos propietarios que aprovechaban esta nueva tecnología, como Doom o Descent, comenzaron a aparecer. Los programadores de juegos libres, siguieron la misma estela y se crearon títulos como Tuxedo T. Penguin: A Quest for Herring by Steve Baker, TuxKart[8], Tuxracer[9] o FlightGear[10], siendo todos ellos claros ejemplos de los primeros juegos tridimensionales libres que se programaron. La aparición de motores gráficos libres como Cube o Genesis3D impulsaron el desarrollo de otros juegos libres en 3D.

Id Software, una empresa estadounidense de desarrollo de videojuegos, fue una de las primeras compañías en desarrollar juegos comerciales para los sistemas Linux y también una de las primeras en colaborar con el software libre al hacer público el código fuente de los juegos de disparo en primera persona Wolfenstein 3D y Doom, primero con una licencia personalizada y posteriormente bajo los términos de la GPL (GNU General Public License).

A estos lanzamientos de código abierto le siguieron los de los motores Quake Engine, id Tech 2 e id Tech 3 (los motores de Quake, Quake II y Quake III respectivamente), que permitieron el nacimiento de nuevos juegos de disparo en primera persona como Freedoom, Nexuiz[11], Tremulous[12] u OpenArena[13].

[8] Videojuego de carreras de karts http://tuxkart.sourceforge.net/
[9] Juego de carreras en el que Tux se desliza ladera abajo http://extremetuxracer.com/
[10] Simulador de vuelo http://www.flightgear.org/
[11] Juego de acción en primera persona http://www.nexuiz.com/
[12] Videojuego de ambientación futurista y acción en primera persona http://tremulous.net/
[13] Juego de disparos en primera persona inspirado en Quake III http://openarena.ws/

Figura 2.9: OpenArena:
Surgido de la liberación del motor de juego de Quake III

Otras compañías que trabajaban en conjunto con id Software, como Raven Software, Bungie Software o 3D Realms, también liberaron el código fuente de juegos como Jump 'n bump, Meritous, Warzone 2100, HoverRace o Abuse, incluso llegaron a hacerlos totalmente libres, incluyendo todos los recursos multimedia y los niveles.

Por otra parte, algunos desarrolladores de juegos de código propietario, han ayudado a la creación de bibliotecas libres para la programación de videojuegos. Por ejemplo, Loki Software ayudó a la creación y mantenimiento de libSDL y la librería de audio OpenAL.

Desde finales de los años 90 hasta nuestros días, muchos programadores y equipos han seguido desarrollando juegos libres (gran parte de ellos clones de versiones comerciales), que han logrado alcanzar una considerable popularidad, como por ejemplo SuperTux[14], FrozenBubble[15], Frets on Fire[16], WarMUX[17], Neverball[18] o SuperTuxKart[19].

[14] Videojuego de plataformas en 2D inspirado en Super Mario Bros http://supertux.lethargik.org/

[15] En FrozenBubble, el jugador controla a Tux, quien se encarga de lanzar bolas de colores para formar grupos de un mismo color que desaparecerán. El objetivo es limpiar todo el nivel de bolas sin que éstas superen una línea que aparece en la parte inferior de la pantalla http://www.frozen-bubble.org/

[16] Juego musical similar a Guitar Hero en el que se simula el acto de tocar una guitarra. Las notas están sincronizadas con la canción y el jugador tiene que pulsar las teclas en el momento justo http://fretsonfire.sourceforge.net/

[17] Juego de artillería y estrategia por turnos, basado en el mítico Worms. El jugador controlará a una grupo de lombrices en un escenario deformable en el que tendrá que derrotar al equipo enemigo http://www.wormux.org/

[18] Videojuego de plataformas en el que el jugador controla a una pelota a la que tendrá que conducir hasta la salida del nivel en un determinado tiempo, similar a Super Monkey Ball http://neverball.org/

[19] Carreras de coche, tipo arcade y en 3D http://supertuxkart.sourceforge.net/

Capítulo 3

Planificación

Para el desarrollo de SDLgolf ha sido necesario emplear varias herramientas, utilidades y bibliotecas.

Algunas de ellas, ya habían sido utilizadas en ciertas asignaturas a lo largo de la carrera. Otras, han requerido un periodo de formación previo, en el que se han adquirido los conocimientos necesarios para poder desarrollar el videojuego.

No obstante, los conceptos y conocimientos ya adquiridos sobre programación y manejo de utilidades, fueron revisados y repasados.

Las primeras fases del desarrollo de SDLgolf estuvieron centradas en el análisis y diseño del sistema y posteriormente se llevó a cabo la codificación.

En primer lugar, se desarrollaron las estructuras básicas del juego que permitían la interacción con la pantalla, para luego dar paso a la programación de la física del sistema.

Seguidamente, se crearon los elementos esenciales para la partida y luego se implementaron los diferentes modos de juego, así como las animaciones y la interfaz.

Por último, se realizaron los diferentes menús y se llevó a cabo la unificación y depuración del código.

3.1. Organización temporal

A continuación, podemos observar el diagrama de desarollo de SDLgolf.

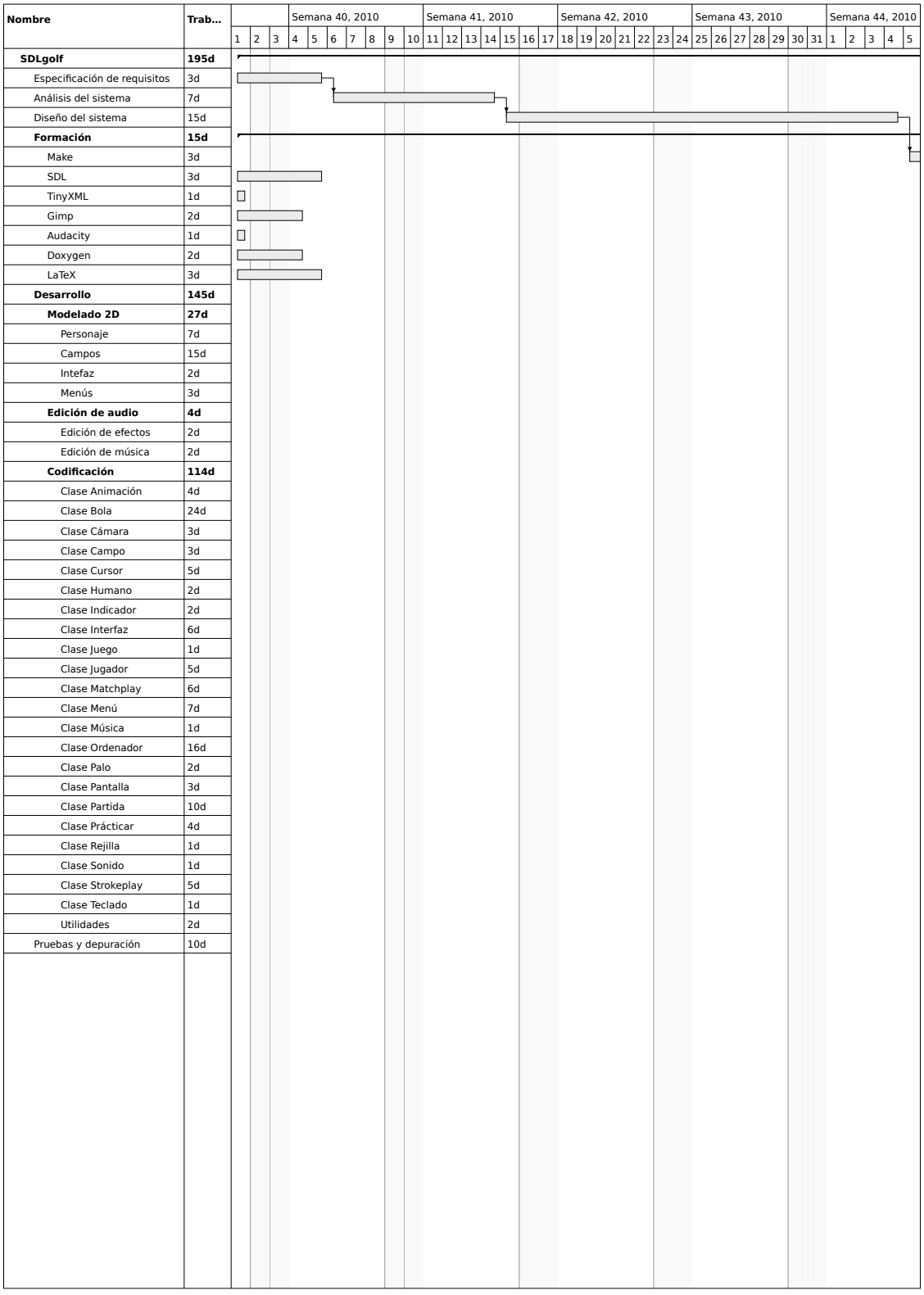


Figura 3.1: Diagrama de Gantt (Parte 1)

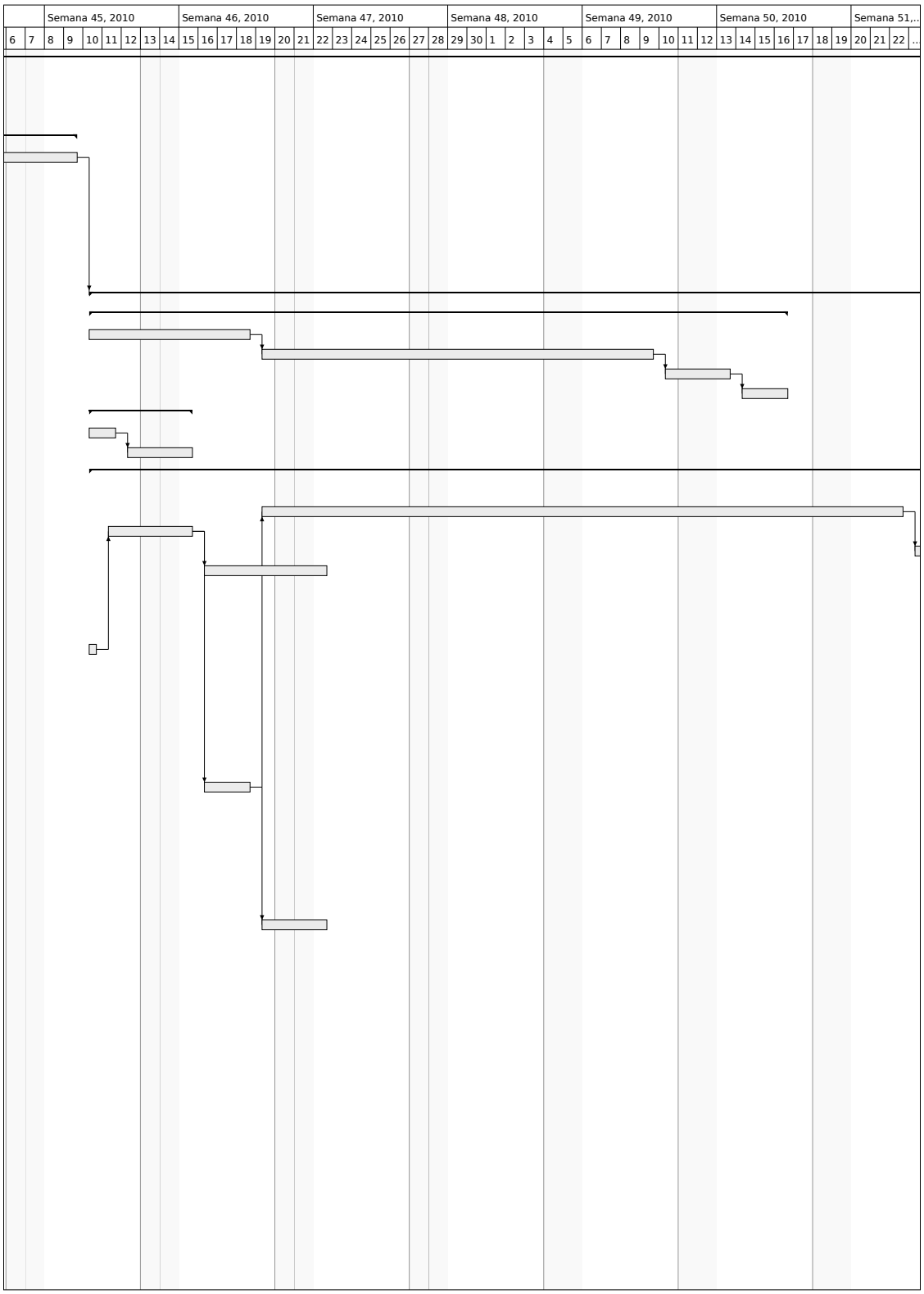


Figura 3.2: Diagrama de Gantt (Parte 2)

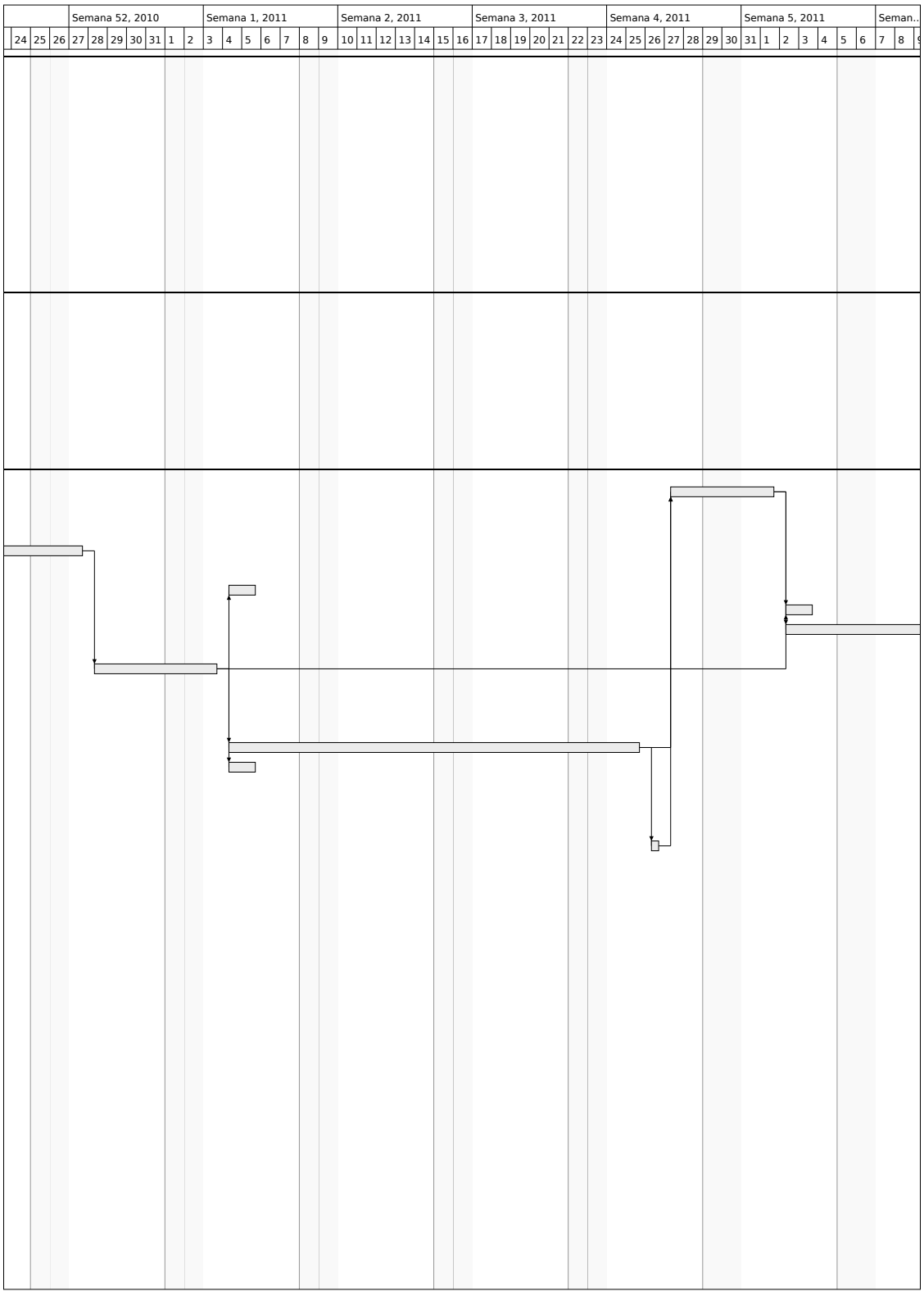


Figura 3.3: Diagrama de Gantt (Parte 3)

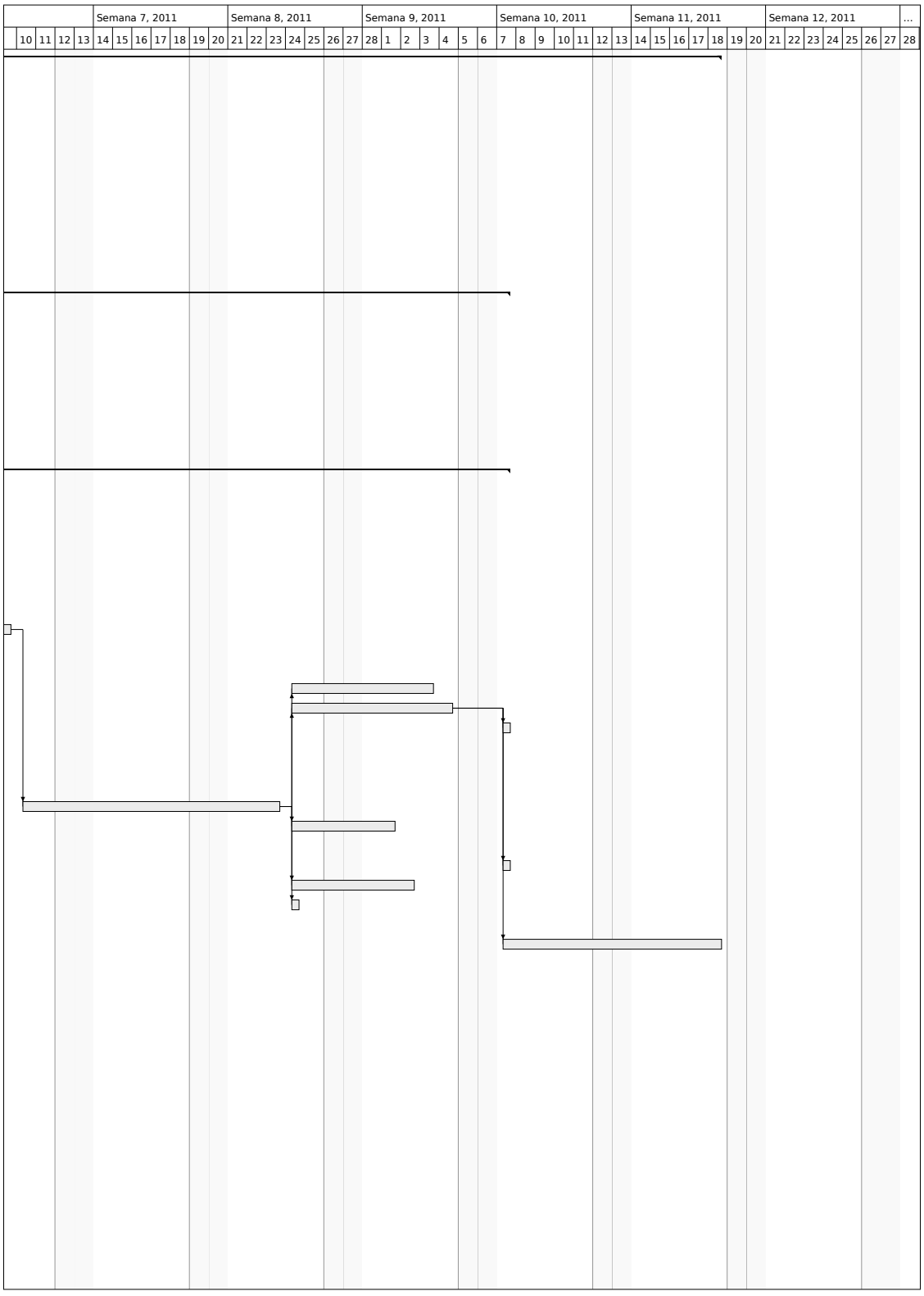


Figura 3.4: Diagrama de Gantt (Parte 4)

Capítulo 4

Especificación de los requisitos del sistema

Para la creación de un producto software, es determinante establecer las condiciones y necesidades que ha de satisfacer el sistema.

El fundamento básico de esta tarea es la recopilación de información. A lo largo de la historia, se han empleado diversas técnicas como las entrevistas o los talleres y más recientemente el prototipado.

Seguiremos un esquema que nos permita describir los requisitos del sistema de una forma metódica y racional.

4.1. Requisitos de interfaces externas

La librería SDL nos proporciona una capa de abstracción del hardware, por lo que no es necesario analizar ni diseñar las características lógicas que tengamos que modificar en los diferentes dispositivos tales como el teclado, el hardware de gráfico, el hardware de sonido, etc.

Para la interacción con el usuario se empleará una ventana no modal[1]a una resolución de 800x480 píxeles, siendo posible establecer el modo a pantalla completa[2]. Se distinguen los siguientes tipos de ventana:

- **Menú principal**. La ventana del menú principal mostrará el logotipo de la aplicación, así como los diferentes modos de juego, las opciones, el menú de información y la opción salir. El usuario podrá interactuar con el menú empleando las teclas de *cursor arriba* y *cursor abajo*. Con la tecla *enter* se podrá confirmar la selección.

[1]Una ventana modal permite cambiar el foco a cualquier otra del entorno gráfico.
[2]El modo a pantalla completa podrá establecerse desde cualquier ventana pulsando la tecla F11.

Figura 4.1: Ventana del menú principal

- **Opciones**. Desde la ventana de opciones podrán desactivarse los efectos de sonido y música o comprobar las teclas asignadas al control del juego. Desde esta ventana será posible regresar a la ventana del menú principal.

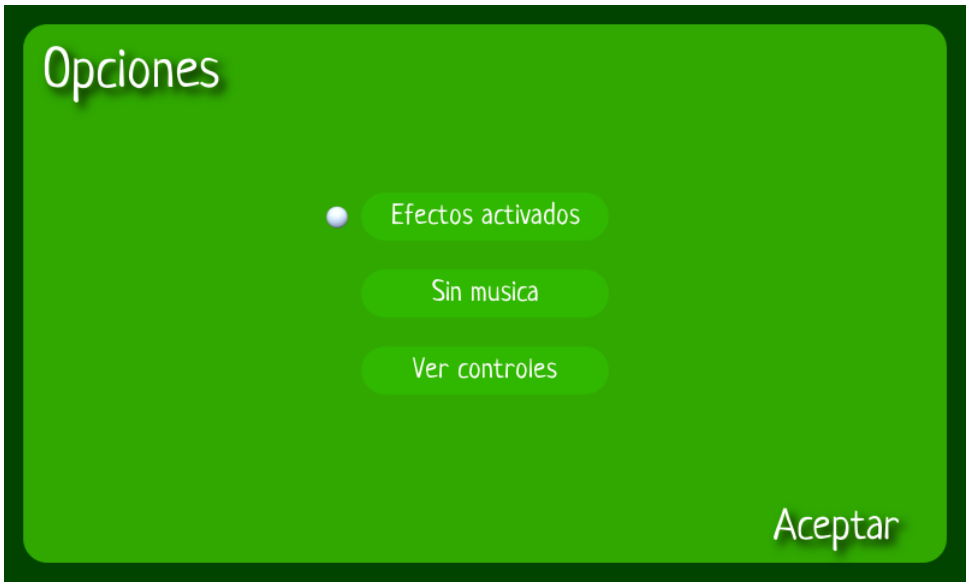


Figura 4.2: Ventana del menú opciones

- **Acerca de**. En esta ventana se mostrará información básica sobre el juego y el autor del mismo.

Figura 4.3: Ventana de información sobre el juego

- **Elección de jugador**. La ventana de elección de jugador será compartida por los diferentes modos de juego. Desde esta ventana se podrá establecer el nombre del jugador mediante la captura de pulsaciones en el teclado y el tipo y color iterando entre los disponibles. También será posible volver a la ventana del menú principal. El usuario podrá moverse entre las diferentes opciones usando las teclas del cursor y la tecla enter para confirmar la selección.

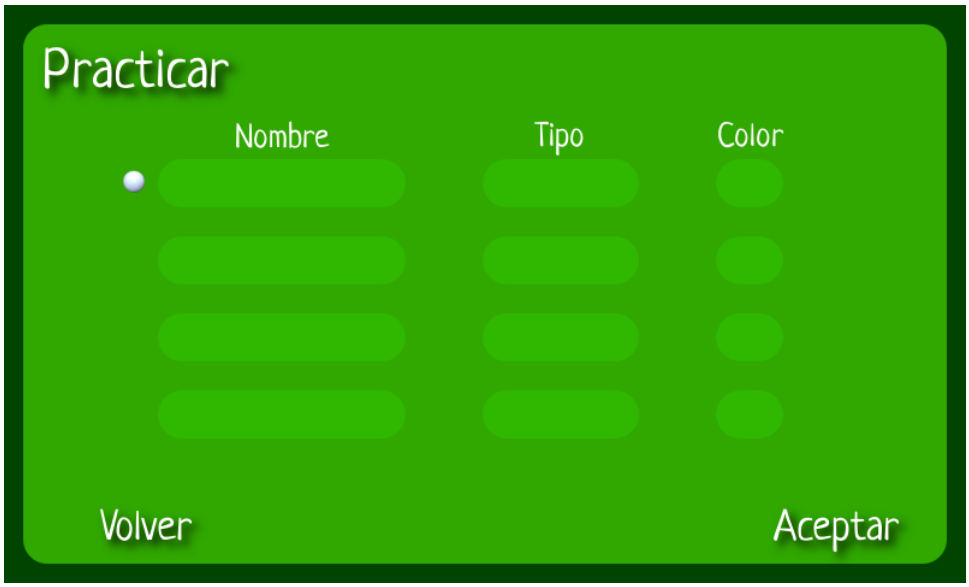


Figura 4.4: Ventana del menú de elección de jugador

- **Elección de campo y hoyo**. La ventana de elección de campo será compartida por los diferentes modos de juego. En esta ventana el usuario podrá elegir el campo

y el número de hoyo en el que desea jugar, pudiendo comenzar la partida. Será posible volver a la ventana de selección de jugador. El usuario podrá moverse entre las diferentes opciones usando las teclas del cursor y la tecla enter para confirmar la selección.

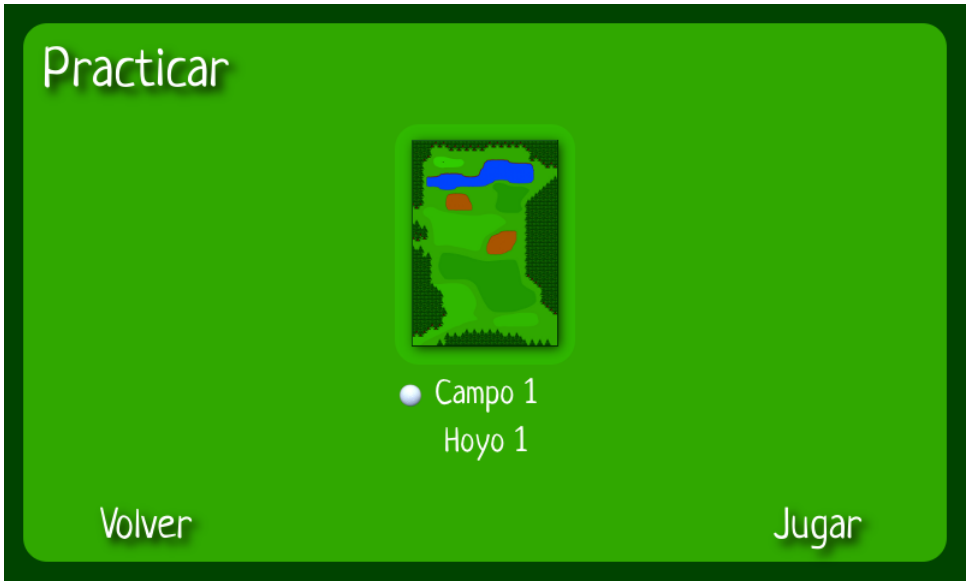


Figura 4.5: Ventana del menú de elección de campo y hoyo

- **Juego**. Ésta será la ventana principal del juego. En ella se representarán todos los elementos gráficos del mismo. La interfaz permitirá al jugador conocer información básica sobre la partida que se esté desarrollando. Será necesario mostrar el nombre y el par del jugador, el nombre del campo actual, el número de hoyo, el par del hoyo, el palo seleccionado actualmente por el jugador, la distancia máxima que alcanza el palo y la distancia existente entre el jugador y el hoyo. Además de esto, será necesario mostrar un cursor que indique la dirección a la que apunta el jugador, así como un indicador de fuerza y efecto aplicado al tiro. El jugador podrá cambiar la dirección de tiro con las teclas cursor izquierda y cursor derecha, el palo con las teclas cursor arriba y cursor abajo. Para realizar el golpe y determinar la fuerza y efecto del mismo se usará la tecla espacio. El modo pausa podrá activarse pulsando la tecla escape.

Figura 4.6: Ventana de juego

- **Pausa**. La ventana de pausa será únicamente accesible desde la ventana de juego. Permitirá poner en pausa la partida en curso, siendo posible continuar jugando o regresar a la ventana del menú principal.

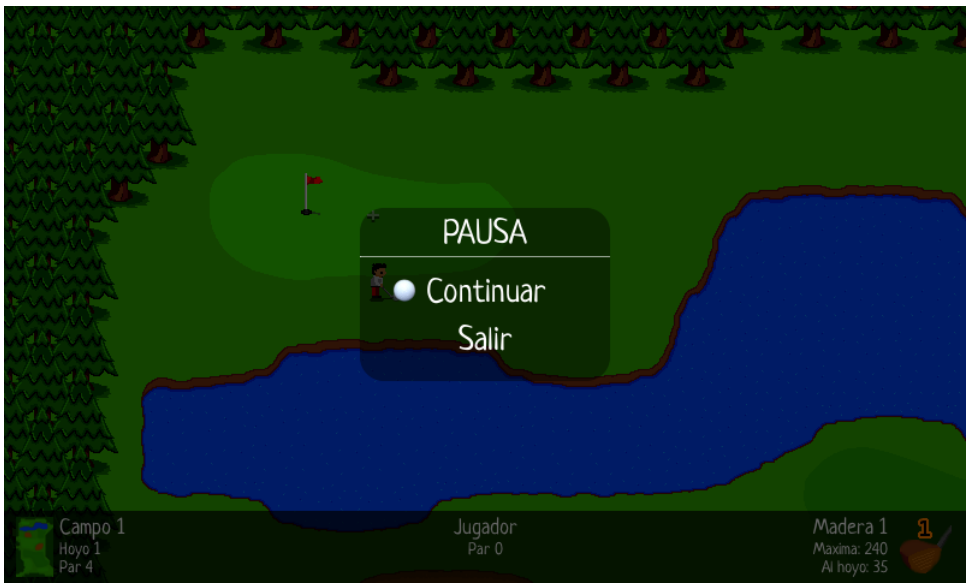


Figura 4.7: Ventana de pausa

- **Final**. Al acabar la partida se mostrará la ventana de finalización de partida. En ella, se indicará el nombre y la puntuación final de los jugadores, así como el nombre del campo, el número del hoyo y el par del mismo. Desde esta ventana, será posible regresar a la ventana del menú principal.

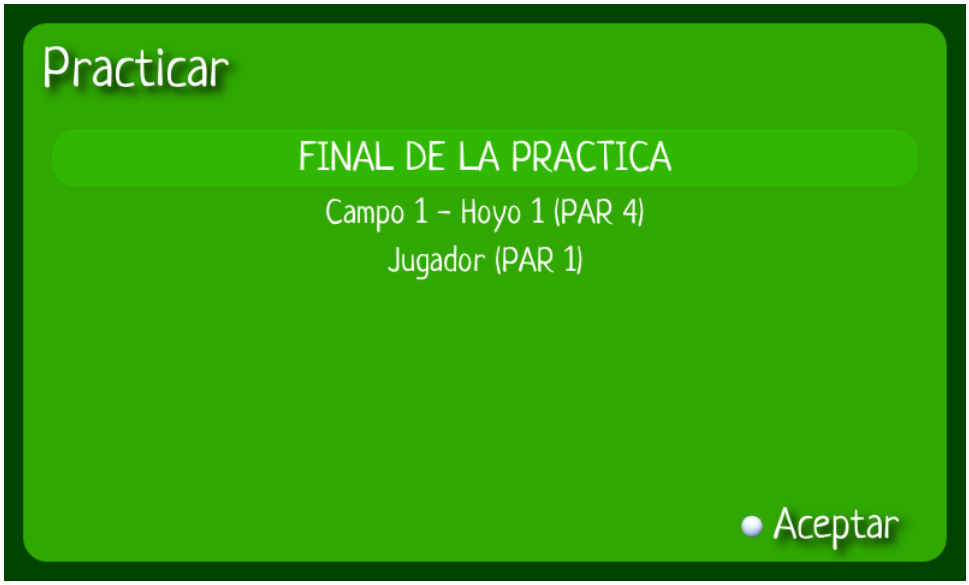


Figura 4.8: Ventana de finalización de partida

4.2. Requisitos funcionales

El comportamiento del software debe cumplir los siguientes requisitos:

- Salir del juego desde cualquier ventana.
- Establecer el modo a pantalla completa desde cualquier ventana.
- Seleccionar los diferentes modos de juego.
- Permitir al jugador enfrentarse al ordenador.
- Establecer, guardar y cargar las opciones del juego.
- Introducir el nombre del jugador desde el teclado.
- Cambiar el tipo y el color del jugador.
- Cambiar el campo y el número de hoyo.
- Pausar la partida.
- Establecer la dirección, fuerza y efecto del tiro.
- Cambiar de palo.
- Gestionar la sucesión de hoyos de un campo durante la partida.

Se definen cuatro tipos de jugadores:

- **Humano**. El jugador es controlado por una persona.
- **IA baja**. El jugador es controlado por el ordenador, procesando el campo con una inteligencia artificial baja.
- **IA media**. El jugador es controlado por el ordenador, procesando el campo con una inteligencia artificial media.
- **IA alta**. El jugador es controlado por el ordenador, procesando el campo con una inteligencia artificial alta.

Se definen tres modos de juego:

- **Practicar**. El jugador seleccionará un hoyo de un determinado campo en el que desee llevar a cabo la práctica. Una vez superado el hoyo, se mostrará la venta de finalización en la que se indicará el par que ha obtenido el jugador y el par que tiene el hoyo del campo en el que ha jugado. Podrán participar hasta cuatro jugadores.
- **Matchplay**. Podrán enfrentarse dos jugadores, dos parejas o una pareja contra un único jugador. Dentro de este modo de juego, se diferencian los siguientes tipos:
  - **Bolas compartidas**. Los miembros de un mismo equipo compartirán la bola de golf y el par.
  - **Bolas independientes**. Cada jugador tendrá su propia bola de golf y el par será también independiente.

  Se elegirá un determinado campo y se completarán todos los hoyos del mismo.

  Al completar un hoyo, el equipo o el jugador que haya obtenido el menor par sumará un punto a su marcador. Si los equipos empatan el par del hoyo, no sumarán puntos. El equipo o el jugador que tenga el mayor par tampoco sumará puntos. Si una equipo está jugando con bolas de golf independientes y uno de los miembros hace hoyo, se comprobará si su compañero puede mejorar el par. En caso de que no pueda mejorarlo, se procederá a finalizar al compañero.

  Ganará la partida el equipo o el jugador que, tras completar todos los hoyos del campo, haya obtenido el mayor número de puntos, es decir, el jugador o el equipo que haya ganado el mayor número de hoyos en un campo.

  Al acabar la partida, se mostrará la ventana de finalización en la que se indicará los puntos totales que han obtenido los jugadores de cada equipo, el nombre del campo y el número de hoyos que tiene el campo.
- **Strokeplay**. En este modo de juego se podrá participar de forma individual o enfrentándose hasta cuatro jugadores.

  Se elegirá un determinado campo y se completarán todos los hoyos del mismo.

  Al completar un hoyo, cada jugador obtendrá como puntuación el resultado de restar al par del propio jugador el par del hoyo.

Ganará la partida el jugador que, tras completar todos los hoyos del campo, haya obtenido la menor puntuación, es decir, el jugador que haya completado todos los hoyos de un campo con el menor número de golpes.

Al acabar la partida, se mostrará la ventana de finalización en la que se indicará los puntos de cada jugador, el nombre del campo y el par del campo.

4.3. Restricciones de rendimiento

La mayor parte del procesamiento recae sobre la física de la bola de golf y los cálculos que realiza la inteligencia artificial. Será necesario optimizar dichos procesos para evitar ralentizaciones en el juego, asegurando que, como mínimo, el juego muestre 60 imágenes por segundo.

De la misma forma, será muy importante optimizar el rendimiento de los procesos que dibujan las imágenes en pantalla, dado que se hará un uso intensivo de ellos.

El sistema deberá responder en un tiempo lógico a los eventos generados por el usuario.

4.4. Restricciones de diseño

El diseño estará basado en la programación orientada a objetos y se dará una mayor prioridad a la optimización temporal frente a la espacial.

La búsqueda del menor tiempo de respuesta determinará el diseño final.

4.5. Atributos del sistema software

Nuestra aplicación deberá cumplir los siguientes atributos:

- Código mantenible y ampliable.
- Portabilidad a otros sistemas compatibles con SDL.
- Fiabilidad en el diseño.
- Robustez en la ejecución.

4.6. Otros requisitos

Se primará la simplicidad en la codificación del juego con la idea de mejorar la comprensión del código y la optimización del mismo.

Capítulo 5

Análisis del sistema

El análisis del sistema nos permite analizar y especificar el comportamiento del sistema. Para ello, emplearemos el lenguaje de modelado de sistemas UML.

5.1. Modelo de casos de uso

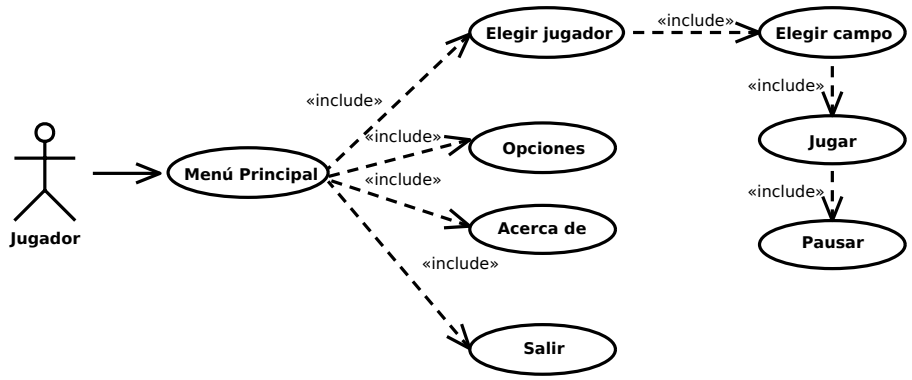


Figura 5.1: Diagrama de casos de uso

5.1.1. Descripción de los casos de uso

Todas las acciones del usuario producirán un efecto de sonido a modo de indicar que el sistema las ha procesado y desde cualquier ventana se podrá volver a la anterior. La notación de estos eventos no se incluirá en las descripciones por una cuestión de redundancia.

5.1.1.1. Caso de uso: Menú principal

- Caso de uso: Menú principal.

- Descripción: Muestra el menú principal de la aplicación, desde donde es posible acceder a los diferentes modos de juego y a las opciones.
- Actores: Usuario, Sistema.
- Precondiciones: Ninguna.
- Postcondiciones: Ninguna.
- Escenario principal.
    1. El sistema muestra el menú principal en la pantalla.
    2. El usuario selecciona el modo Practicar.
    3. El sistema inicia el modo de elección de jugador.
- Extensiones (Flujo alternativo):

    ***a.** El usuario cierra la ventana del juego.
        1. El sistema sale de la aplicación.

    ***b.** El usuario pulsa la tecla de pantalla completa.
        1. El sistema establece el modo a pantalla completa.

    2a. El usuario selecciona el modo Matchplay.
        1. El sistema inicia el modo elección de jugador.

    2b. El usuario selecciona el modo Strokeplay.
        1. El sistema inicia el modo elección de jugador.

    2c. El usuario selecciona las opciones.
        1. El sistema inicia las opciones.

    2d. El usuario selecciona información sobre el juego.
        1. El sistema inicia la ventana de información.

    2e. El usuario selecciona salir.
        1. El sistema sale de la aplicación.

5.1.1.2. Caso de uso: Elegir jugador

- Caso de uso: Elegir jugador.
- Descripción: Muestra el menú de elección de jugador desde donde el usuario puede elegir el nombre, tipo y color del jugador.
- Actores: Usuario, Sistema.
- Precondiciones: Ninguna.

- Postcondiciones: Se crea al jugador.
- Escenario principal.

  1. El usuario ha elegido el modo Practicar.

  1. El sistema muestra el menú de elección de jugador.
  2. El usuario presiona enter en la casilla del nombre del jugador.
  3. El sistema captura un máximo de caracteres introducidos desde el teclado.
  4. El usuario introduce el nombre del jugador y vuelve a presionar enter para finalizar la introducción de texto.
  5. El sistema establece el tipo de jugador a humano y un color aleatorio.
  6. El usuario confirma los datos.
  7. El sistema inicia el modo de elección de campo.

- Extensiones (Flujo alternativo):

  ***a.** El usuario cierra la ventana del juego.

  1. El sistema sale de la aplicación.

  ***b.** El usuario pulsa la tecla de pantalla completa.

  1. El sistema establece el modo a pantalla completa.

  ***c.** El usuario selecciona volver.

  1. El sistema vuelve a la ventana anterior.

  2b. El usuario presiona la tecla espacio en la casilla del nombre del jugador.

  1. El sistema genera un nombre y un color aleatorio y establece el tipo de jugador a humano.
  2. El usuario confirma los datos.
  3. El sistema inicia el modo de elección de campo.

  5a. El usuario presiona la tecla escape.

  1. El sistema cancela la captura de texto y vuelve al paso 1.

  5b. El usuario introduce un nombre vacío.

  1. El sistema cancela la captura de texto y vuelve al paso 1.

  6a. El usuario presiona la tecla enter en la casilla del nombre.

  1. El sistema permite editar el nombre y captura un máximo de caracteres introducidos desde el teclado.
  2. El usuario introduce el nombre del jugador y presiona enter para finalizar la introducción de texto.

2a. El usuario presiona la tecla escape.

1. El sistema cancela la captura de texto y vuelve al paso 1.

2b. El usuario introduce un nombre vacío.

1. El sistema cancela la captura de texto y vuelve al paso 1.

3. El sistema vuelve al paso 5.

6b. El usuario presiona la tecla de borrar en la casilla del nombre del jugador.

1. El sistema elimina al jugador y vuelve al paso 1.

6c. El usuario presiona la tecla enter en la casilla del tipo.

1. El sistema selecciona el siguiente tipo para el jugador y vuelve al paso 6.

6d. El usuario presiona la tecla enter en la casilla del color.

1. El sistema selecciona el siguiente color para el jugador y vuelve al paso 6.

1a. El usuario ha elegido el modo Matchplay.

1. El sistema muestra el menú de elección de jugador.
2. El usuario presiona enter en la casilla del nombre del primer jugador.
3. El sistema captura un máximo de caracteres introducidos desde el teclado.
4. El usuario introduce el nombre del jugador y vuelve a presionar enter para finalizar la introducción de texto.
5. El sistema establece el tipo de jugador a humano y un color aleatorio.
6. El usuario presiona enter en la casilla del nombre del segundo jugador.
7. El sistema captura un máximo de caracteres introducidos desde el teclado.
8. El usuario introduce el nombre del segundo jugador y vuelve a presionar enter para finalizar la introducción de texto.
9. El sistema establece el tipo de jugador a humano y un color aleatorio.
10. El usuario confirma los datos.

    10a. El usuario presiona la tecla espacio en la casilla del nombre del tercer jugador.

    1. El sistema captura un máximo de caracteres introducidos desde el teclado.
    2. El usuario introduce el nombre del tercer jugador y vuelve a presionar enter para finalizar la introducción de texto.
    3. El sistema establece el tipo de jugador a humano y un color aleatorio.
    4. El usuario confirma los datos.
    5. El sistema muestra las parejas disponibles para el primer jugador.
    6. El usuario elige la pareja del primer jugador.

7. El sistema inicia el modo de elección de campo.

11. El sistema inicia el modo de elección de campo.

1b. El usuario ha elegido el modo Strokeplay.

1. El sistema muestra el menú de elección de jugador.
2. El usuario presiona enter en la casilla del nombre del jugador.
3. El sistema captura un máximo de caracteres introducidos desde el teclado.
4. El usuario introduce el nombre del jugador y vuelve a presionar enter para finalizar la introducción de texto.
5. El sistema establece el tipo de jugador a humano y un color aleatorio.
6. El usuario confirma los datos.
7. El sistema inicia el modo de elección de campo.

5.1.1.3. Caso de uso: Elegir campo

- Caso de uso: Elegir campo.
- Descripción: Muestra el menú de elección de campo desde donde el usuario puede elegir el campo y el número de hoyo en el que desea jugar.
- Actores: Usuario, Sistema.
- Precondiciones: Existe al menos un jugador.
- Postcondiciones: Se cargan los datos del campo.
- Escenario principal.

  1. El usuario ha elegido el modo Practicar.

  1. El sistema muestra el menú de elección de campo, seleccionando el primer hoyo del primer campo.
  2. El usuario confirma los datos.
  3. El sistema inicia la partida de práctica.

- Extensiones (Flujo alternativo):

  ***a.** El usuario cierra la ventana del juego.

  1. El sistema sale de la aplicación.

  ***b.** El usuario selecciona volver.

  1. El sistema vuelve a la ventana anterior.

  2a. El usuario selecciona el siguiente campo.

1. El sistema muestra el primer hoyo del campo elegido.
2. El usuario confirma los datos.
3. El sistema inicia la partida de práctica.

2b. El usuario selecciona el siguiente hoyo.

1. El usuario confirma los datos.
2. El sistema inicia la partida de práctica.

1a. El usuario ha elegido el modo Matchplay.

1. El sistema muestra el menú de elección de campo, seleccionando el primer campo.
2. El usuario confirma los datos.
3. El sistema inicia el modo de juego Matchplay.

1b. El usuario ha elegido el modo Strokeplay.

1. El sistema muestra el menú de elección de campo, seleccionando el primer campo.
2. El usuario confirma los datos.
3. El sistema inicia el modo de juego Strokeplay.

5.1.1.4. Caso de uso: Jugar

- Caso de uso: Jugar.
- Descripción: El sistema comienza la partida y muestra la pantalla de juego.
- Actores: Usuario, Sistema.
- Precondiciones: Ninguna.
- Postcondiciones: Ninguna.
- Escenario principal.

  1. El usuario ha elegido el modo de juego Practicar.

  1. El sistema muestra la pantalla de juego.
  2. El jugador presiona la tecla espacio.
  3. El sistema muestra el nivel de fuerza.
  4. El jugador vuelve a presionar la tecla espacio para confirmar el nivel de fuerza.
  5. El sistema muestra el nivel de efecto.
  6. El jugador vuelve a presionar la tecla espacio para confirmar el nivel de efecto.
  7. El sistema realiza el tiro y la bola hace hoyo.

8. El sistema finaliza la partida y muestra una pantalla de resumen.
9. El usuario confirma la lectura del resumen.
10. El sistema vuelve al menú principal.

- Extensiones (Flujo alternativo):

  ***a.** El usuario cierra la ventana del juego.
    1. El sistema sale de la aplicación.

  2a. El usuario presiona la tecla cursor arriba.
    1. El sistema selecciona el palo siguiente y vuelve al paso 1.

  2b. El usuario presiona la tecla cursor abajo.
    1. El sistema selecciona el palo anterior y vuelve al paso 1.

  2c. El usuario presiona la tecla cursor izquierda.
    1. El sistema gira el cursor a la izquierda y vuelve al paso 1.

  2d. El usuario presiona la tecla cursor derecha y vuelve al paso 1.
    1. El sistema gira el cursor a la derecha.

  2e. El usuario presiona la tecla del mapa del campo.
    1. El sistema muestra el mapa del campo y vuelve al paso 1.

  2f. El usuario presiona la tecla de pausa.
    1. El sistema inicia del modo pausa.

  2g. El jugador activo es el ordenador.
    1. El ordenador realiza el tiro y hace hoyo.

  3a. El usuario presiona la tecla escape.
    1. El sistema cancela el tiro y vuelve al paso 1.

  7a. El sistema realiza el tiro y la bola no hace hoyo.
    1. El sistema vuelva al paso 1.

  1a. El usuario ha elegido el modo de juego Matchplay.
    1. El sistema muestra la pantalla de juego.
    2. El primer jugador presiona la tecla espacio.
    3. El sistema muestra el nivel de fuerza.
    4. El primer jugador vuelve a presionar la tecla espacio para confirmar el nivel de fuerza.
    5. El sistema muestra el nivel de efecto.

6. El jugador vuelve a presionar la tecla espacio para confirmar el nivel de efecto.
7. El sistema realiza el tiro y la bola hace hoyo.

   6a. El sistema realiza el tiro y la bola no hace hoyo.

   a) El sistema desactiva al primer jugador, activa al segundo y continúa con el paso 9.
8. El sistema finaliza al primer jugador y activa al segundo.
9. El segundo jugador presiona la tecla espacio.
10. El sistema muestra el nivel de fuerza.
11. El segundo jugador vuelve a presionar la tecla espacio para confirmar el nivel de fuerza.
12. El sistema muestra el nivel de efecto.
13. El segundo jugador vuelve a presionar la tecla espacio para confirmar el nivel de efecto.
14. El sistema realiza el tiro y la bola hace hoyo.

    14a. El sistema realiza el tiro y la bola no hace hoyo.

    1. El sistema vuelva al paso 9.
15. El sistema finaliza la partida y muestra una pantalla de resumen.

    15a. El sistema carga el siguiente hoyo del campo.

    1. El sistema vuelve al paso 1.
16. El usuario confirma la lectura del resumen.
17. El sistema vuelve al menú principal.

1b. El usuario ha elegido el modo de juego Strokeplay.

1. El sistema muestra la pantalla de juego.
2. El jugador presiona la tecla espacio.
3. El sistema muestra el nivel de fuerza.
4. El jugador vuelve a presionar la tecla espacio para confirmar el nivel de fuerza.
5. El sistema muestra el nivel de efecto.
6. El jugador vuelve a presionar la tecla espacio para confirmar el nivel de efecto.
7. El sistema realiza el tiro y la bola hace hoyo.

   7a. El sistema realiza el tiro y la bola no hace hoyo.

   1. El sistema vuelve al paso 1.
8. El sistema finaliza la partida y muestra una pantalla de resumen.

   8a. El sistema carga el siguiente hoyo del campo.

   1. El sistema vuelve al paso 1.
9. El usuario confirma la lectura del resumen.
10. El sistema vuelve al menú principal.

5.1.1.5. Caso de uso: Pausar

- Caso de uso: Pausar.
- Descripción: Establece el modo de pausa en la partida en curso.
- Actores: Usuario, Sistema.
- Precondiciones: Existe una partida en juego.
- Postcondiciones: Ninguna.
- Escenario principal.
    1. El sistema muestra el menú de pausa.
    2. El usuario selecciona continuar cuando esté preparado.
    3. El sistema reanuda la partida.
- Extensiones (Flujo alternativo):

    ***a.** El usuario cierra la ventana del juego.
        1. El sistema sale de la aplicación.

    2a. El usuario selecciona salir.
        1. El sistema vuelve al menú principal.

5.1.1.6. Caso de uso: Opciones

- Caso de uso: Opciones.
- Descripción: El usuario desea establecer las opciones del juego.
- Actores: Usuario, Sistema.
- Precondiciones: Ninguna.
- Postcondiciones: Ninguna.
- Escenario principal.
    1. El sistema muestra el menú de opciones.
    2. El usuario acepta las opciones.
    3. El sistema vuelve al menú principal.
- Extensiones (Flujo alternativo):

    ***a.** El usuario cierra la ventana del juego.
        1. El sistema sale de la aplicación.

2a. El usuario activa el sonido.
    1. El sistema activa el sonido.

2b. El usuario desactiva el sonido.
    1. El sistema desactiva el sonido.

2c. El usuario activa la música.
    1. El sistema activa la música.

2d. El usuario desactiva la música.
    1. El sistema desactiva la música.

2e. El usuario selecciona mostrar los controles.
    1. El sistema muestra los controles del juego.

5.1.1.7. Caso de uso: Acerca de

- Caso de uso: Acerca de.
- Descripción: El usuario desea obtener información sobre el juego.
- Actores: Usuario, Sistema.
- Precondiciones: Ninguna.
- Postcondiciones: Ninguna.
- Escenario principal.
    1. El sistema muestra información sobre el juego.
    2. El usuario acepta la información.
    3. El sistema vuelve al menú principal.
- Extensiones (Flujo alternativo):

*__a.__ El usuario cierra la ventana del juego.
    1. El sistema sale de la aplicación.

5.1.1.8. Caso de uso: Salir

- Caso de uso: Salir.
- Descripción: El usuario quiere salir del juego.
- Actores: Usuario, Sistema.
- Precondiciones: Ninguna.
- Postcondiciones: Ninguna.
- Escenario principal.
    1. El sistema sale de la aplicación.

5.2. Modelo conceptual de datos

5.2.1. Diagramas de clases conceptuales

Para el desarrollo del juego necesitaremos, principalmente, las siguientes clases:

- Animación: Animación de imágenes.
- Bola: Bola de golf.
- Cámara: Cámara de juego.
- Campo: Campo de golf.
- Cursor: Cursor de dirección.
- Humano: Jugador controlado por una persona.
- Indicador: Indicador de fuerza y efecto.
- Interfaz: Interfaz del juego.
- Juego: Clase principal.
- Jugador: Clase virtual.
- Matchplay: Modo de juego Matchplay.
- Menú: Menú del juego.
- Música: Música del juego.
- Ordenador: Jugador controlado por el ordenador.
- Palo: Palo de golf.
- Pantalla: Pantalla del juego.
- Partida: Clase virtual.
- Practicar: Modo de juego de práctica.
- Rejilla: Rejilla de imágenes.
- Sonido: Efectos de sonido.
- Strokeplay: Modo de juego Strokeplay.
- Teclado: Teclado del juego.

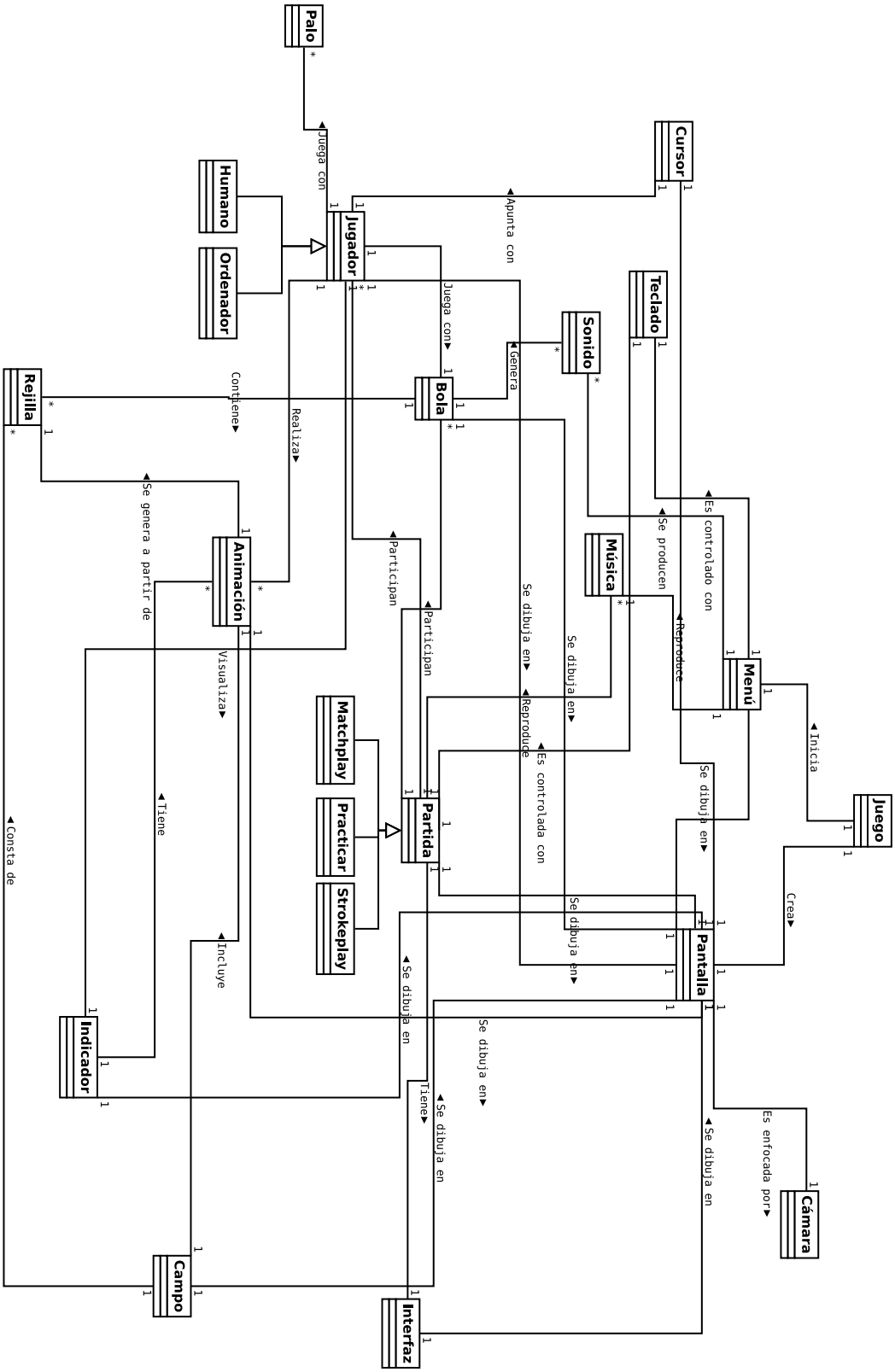


Figura 5.2: Modelo conceptual de datos

5.2.2. Modelo de comportamiento del sistema

5.2.2.1. Modelo de comportamiento: Menú principal

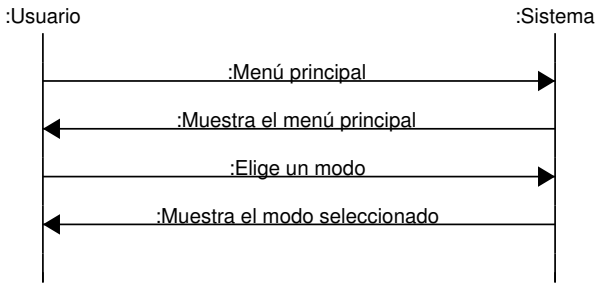


Figura 5.3: Diagrama de secuencia para el caso de uso Menú principal

Contrato de operaciones

- **Operación**: Menú principal.
- **Responsabilidades**: Muestra el menú principal desde donde se pueden elegir los modos de juego, las opciones, información o salir.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Ninguna.

5.2.2.2. Modelo de comportamiento: Elegir jugador

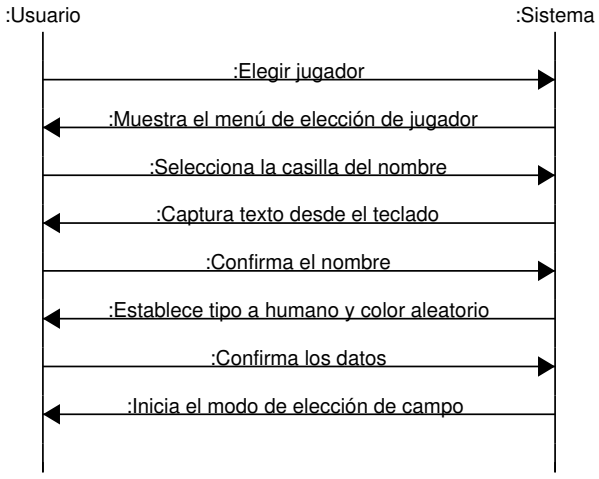


Figura 5.4: Diagrama de secuencia para el caso de uso Elegir jugador

Contrato de operaciones

- **Operación**: Elegir jugador.

- **Responsabilidades**: Muestra el menú de elección de jugador desde donde el usuario puede determinar el nombre, tipo y color del jugador.

- **Precondiciones**: Ninguna.

- **Postcondiciones**: Crea al jugador.

5.2.2.3. Modelo de comportamiento: Elegir campo

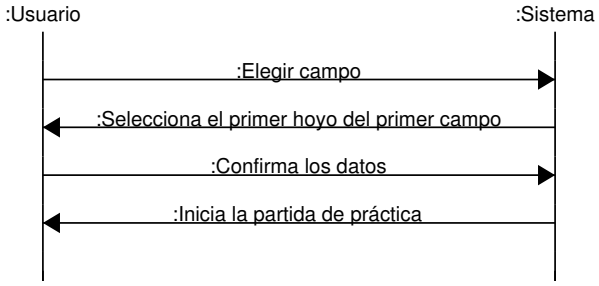


Figura 5.5: Diagrama de secuencia para el caso de uso Elegir campo

Contrato de operaciones

- **Operación**: Elegir campo.

- **Responsabilidades**: Muestra el menú de elección de campo, desde donde el usuario puede seleccionar el campo y el hoyo en el que desea jugar.

- **Precondiciones**: Existe al menos un jugador.

- **Postcondiciones**: Crea el campo de juego.

5.2.2.4. Modelo de comportamiento: Jugar

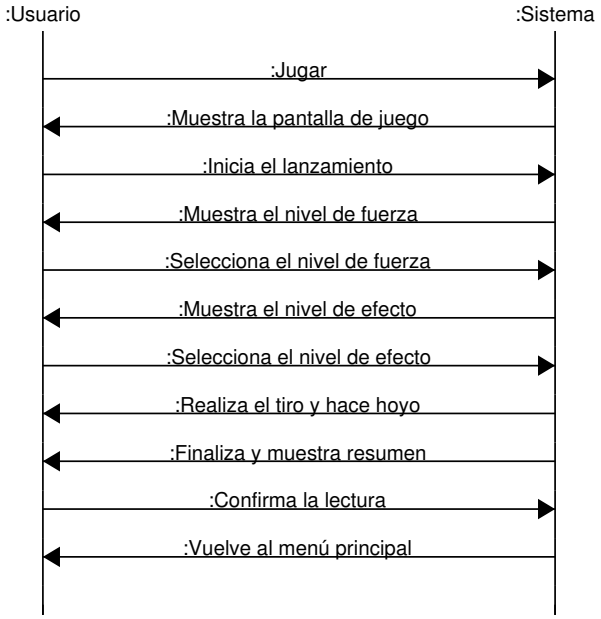


Figura 5.6: Diagrama de secuencia para el caso de uso Jugar

Contrato de operaciones

- **Operación**: Jugar.
- **Responsabilidades**: Muestra la pantalla de juego desde la que el usuario puede jugar la partida.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Ninguna.

5.2.2.5. Modelo de comportamiento: Pausar

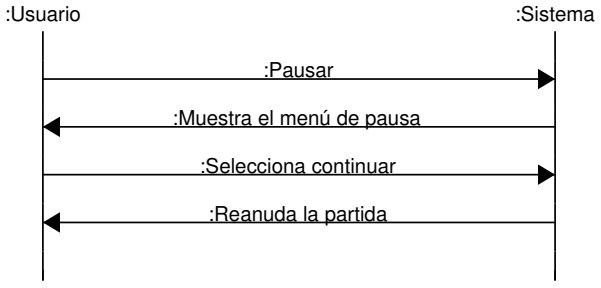


Figura 5.7: Diagrama de secuencia para el caso de uso Pausar

Contrato de operaciones

- **Operación**: Pausar.
- **Responsabilidades**: Muestra el menú de pausa y detiene la partida. Reanuda la partida cuando el usuario esté preparado.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Ninguna.

5.2.2.6. Modelo de comportamiento: Opciones

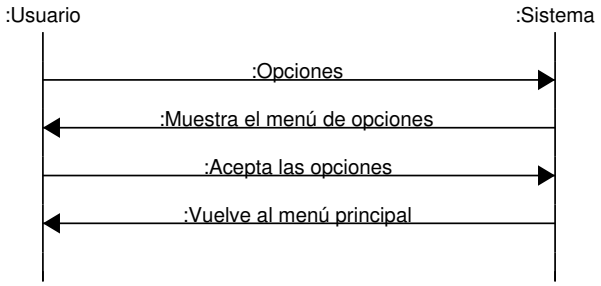


Figura 5.8: Diagrama de secuencia para el caso de uso Opciones

Contrato de operaciones

- **Operación**: Opciones.
- **Responsabilidades**: Muestra las opciones del juego desde donde el jugador puede activar o desactivar el sonido y la música y ver los controles del juego.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Ninguna.

5.2.2.7. Modelo de comportamiento: Acerca de

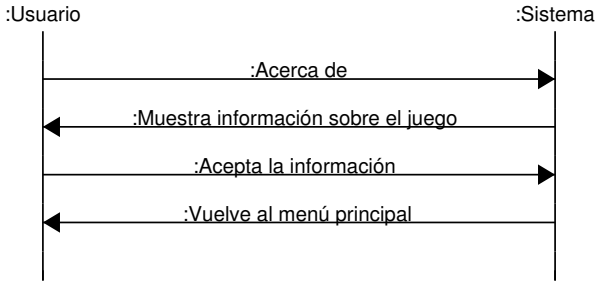


Figura 5.9: Diagrama de secuencia para el caso de uso Acerca de

Contrato de operaciones

- **Operación**: Acerca de.
- **Responsabilidades**: Muestra información sobre el juego.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Ninguna.

5.2.2.8. Modelo de comportamiento: Salir

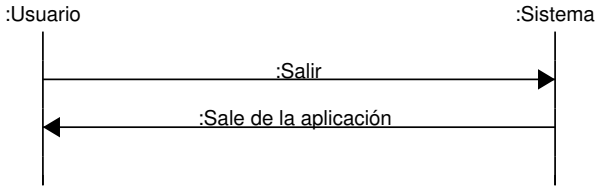


Figura 5.10: Diagrama de secuencia para el caso de uso Salir

Contrato de operaciones

- **Operación**: Salir.
- **Responsabilidades**: Permite al usuario salir del juego.
- **Precondiciones**: Ninguna.
- **Postcondiciones**: Ninguna.

Capítulo 6

Diseño del sistema

Para el desarrollo de SDLgolf, seguiremos una arquitectura basada en capas. Por una parte, la capa de datos nos permitirá cargar de forma dinámica los datos relacionados con el campo de juego. Por otra parte, la capa de dominio implementará todas las funcionalidades del sistema software. Y por último, la capa de presentación, se encargará de la representación de la interfaz gráfica y todos los elementos gráficos del juego.

6.1. Capa de gestión de datos

Con la intención de que SDLgolf sea fácilmente ampliable, es necesario dotar al sistema de una arquitectura que nos permita cargar los datos de los diferentes campos de forma dinámica. Para ello, se emplearán ficheros XML en los que se incorporará toda la información necesaria para la agregación del nuevo campo a nuestro juego.

Los documentos XML, se caracterizan por su fácil comprensión, por su capacidad de hacerse extensibles añadiéndoles nuevas etiquetas y por la flexibilidad que nos brinda al dotar de significado a la información. Además de esto, el analizador XML es un componente estándar, lo que nos permite usar cualquiera de ellos acelerando así el desarrollo de nuestra aplicación.

Nosotros nos decantaremos por TinyXML, por ser un pequeño, simple y potente analizador sintáctico de código abierto.

Para más información sobre la estructura de los documentos XML que emplearemos en SDLgolf consulte el apéndice Formato de los documentos XML.

6.2. Capa de dominio

6.2.1. Clase Animación

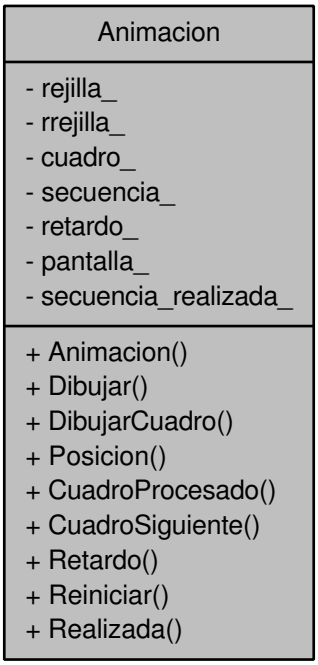


Figura 6.1: Detalle de la clase Animación en UML

6.2.1.1. Descripción

Animación de imágenes.

Esta clase implementa la generación de una animación mediante la representación de una determinada secuencia de imágenes extraídas de una rejilla. El efecto de animación se consigue por la superposición de imágenes en un intervalo de tiempo.

6.2.1.2. Documentación del constructor y destructor

Animacion::Animacion (Rejilla $*$ ***rejilla*****, const int** $*$ ***secuencia*****, const Uint32 & *retardo*, Pantalla** $*$ ***pantalla*****)** Constructor.

Constructor de la animación.

Realiza las asociaciones con la pantalla y la rejilla. Copia la secuencia de la animación y establece el retardo de la misma.

Parámetros:

rejilla Rejilla de imágenes a partir de la cual se generará la animación.

secuencia Secuencia de números que representan los cuadros de la rejilla con los que se generará la animación.

retardo Retardo en milisegundos que habrá entre la representación de cada cuadro de la animación.

pantalla Pantalla en la que se dibujará la animación.

6.2.1.3. Documentación de las funciones miembro

int Animacion::CuadroProcesado () const

Cuadro dibujado.

Devuelve: Último cuadro de la animación que ha sido dibujado.

int Animacion::CuadroSiguiente () const

Cuadro a dibujar.

Devuelve: Cuadro de la animación que se va a dibujar.

void Animacion::Dibujar ()

Dibuja el cuadro y avanza en la secuencia.

Dibuja el cuadro de la animación en la pantalla y selecciona el siguiente cuadro de la animación.

void Animacion::DibujarCuadro ()

Dibuja el cuadro sin avanzar.

Dibuja el cuadro actual de la animación en la pantalla sin cambiar el cuadro de la animación.

void Animacion::Posicion (const SDL_Rect & *posicion*)

Cambia la posición.

Cambia la posición de la animación.

Parámetros:

posicion Posición en la que se dibujará la animación.

bool Animacion::Realizada () const

Animación dibujada completamente.

Devuelve:

- **Verdadero** En caso de haberse realizado la secuencia de animación.
- **Falso** En caso contrario.

void Animacion::Reiniciar ()

Reinicia la secuencia.

Reinicia la animación al principio de la secuencia.

Uint32 Animacion::Retardo () const

Retardo de la animación.

Devuelve: Retardo en milisegundos que habrá entre la representación de cada cuadro de la animación.

6.2.1.4. Documentación de los datos miembro

int Animacion::cuadro_ `[private]`

Cuadro seleccionado de la rejilla de animación.

Pantalla* Animacion::pantalla_ `[private]`

Pantalla del juego.

Rejilla* Animacion::rejilla_ `[private]`

Rejilla de imágenes de la animación.

Uint32 Animacion::retardo_ `[private]`

Retardo en milisegundos que habrá entre la representación de cada cuadro de la animación.

SDL_Rect Animacion::rrejilla_ `[private]`

Posición de la rejilla de imágenes.

std::vector<int> Animacion::secuencia_ `[private]`

Secuencia de la animación.

bool Animacion::secuencia_realizada_ `[private]`
Indica si la animación se ha realizado.

6.2.2. Clase Bola

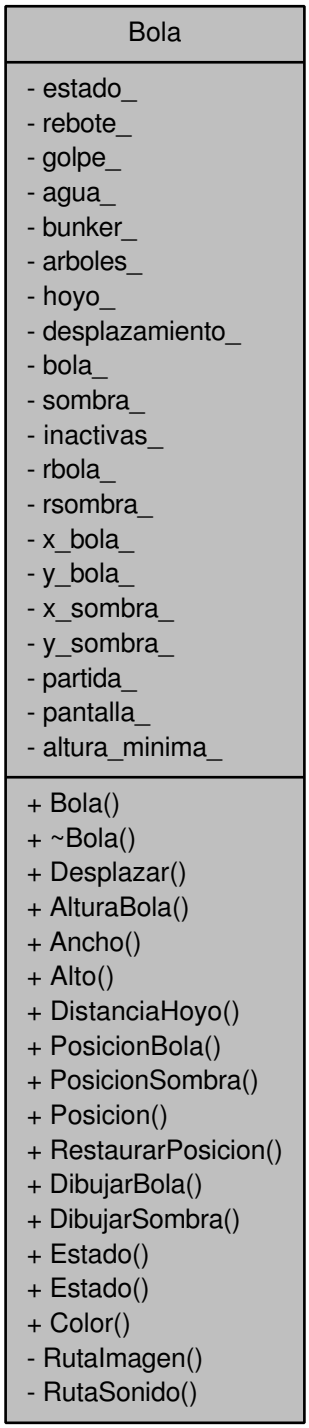


Figura 6.2: Detalle de la clase Bola en UML

6.2.2.1. Descripción

Esta es una de las clases más importantes del juego.

Se encarga de la implementación de la física de la bola de golf y realiza todos los cálculos necesarios para simular el desplazamiento de dicha bola sobre la superficie de juego.

6.2.2.2. Documentación de las enumeraciones miembro de la clase

enum Bola::Estados

Estados de la bola.

Estados en los que puede encontrarse la bola.

Valores de la enumeración:

ACTIVA La bola está actualmente en juego.

INACTIVA La bola está a la espera de entrar en juego.

BUNKER La bola está ACTIVA y además se encuentra en un bunker.

BUNKER_INACTIVA La bola está INACTIVA y además se encuentra en un bunker.

FINALIZADA La bola ha entrado en el hoyo del campo.

6.2.2.3. Documentación del constructor y destructor

Bola::Bola (Partida $*$ *partida*, Pantalla $*$ *pantalla*)

Constructor.

Constructor de la bola de golf.

Realiza las asociaciones con la pantalla y la partida. Se cargan las imágenes de la bola, la sombra y la rejilla de bolas inactivas.

De la misma forma, se cargan los efectos de sonido que produce la bola al caer al agua, al golpear contra el bunker o al caer en los árboles.

El estado inicial de la bola es inactivo.

Parámetros:

partida Partida en curso.

pantalla Pantalla donde se dibujará la bola.

Bola::$\sim$Bola ()

Destructor.

Destructor de la bola.

Destruye las imágenes y la rejilla de bolas inactivas.

6.2.2.4. Documentación de las funciones miembro

int Bola::Alto () const

Alto de la bola.

Alto de la imagen de la bola.

Devuelve: El alto de la imagen de la bola.

int Bola::AlturaBola () const

Altura de la bola.

Calcula la altura a la que se encuentra la bola. La altura viene dada por la diferencia en el *eje de las y* entre la imagen de la bola y su sombra.

Devuelve: La altura a la que se encuentra la bola con respecto al suelo.

int Bola::Ancho () const

Ancho de la bola.

Ancho de la imagen de la bola.

Devuelve: El ancho de la imagen de la bola.

void Bola::Color (int *color*)

Cambia el color.

Establece el color de la bola inactiva.

Parámetros: ***color*** Color que tendrá la bola inactiva.

int Bola::Desplazar (int *x*, int *y*)

Desplaza la bola.

Realiza el desplazamiento de la bola hasta las coordenadas (x, y).

La bola llegará hasta dichas coordenadas y si tiene suficiente inercia rebotará o rodará.

Parámetros:

x Coordenada x hasta la que se desplazará la bola.

y Coordenada y hasta la que se desplazará la bola.

palo Palo con el que se golpea la bola.

Devuelve: **0** cuando el lanzamiento ha finalizado.

void Bola::DibujarBola () const

Dibuja la bola.

Dibuja la bola en la pantalla.

void Bola::DibujarSombra () const

Dibuja la sombra.

Dibuja la sombra en la Pantalla.

int Bola::DistanciaHoyo (const SDL_Rect & *hoyo*) const

Distancia al hoyo.

La función calcula la distancia euclídea entre la posición de la bola y el hoyo del campo.

Parámetros: ***hoyo*** Localización del hoyo.

Devuelve: La distancia en línea recta que hay desde la bola al hoyo.

void Bola::Estado (int *estado*)

Cambia el estado.

Establece un nuevo estado para la bola.

Parámetros: ***estado*** Estado que pasará a tener la bola.

int Bola::Estado () const

Estado de la bola.

Devuelve: Estado actual de la bola.

void Bola::Posicion (const SDL_Rect & *posicion*)

Cambiar posición.

Establece una nueva posición para la bola.

Parámetros: ***posicion*** Nueva posición de la bola.

const SDL_Rect & Bola::PosicionBola () const

Rectángulo de posición de la bola.

Devuelve: Rectángulo de posición de la bola.

const SDL_Rect & Bola::PosicionSombra () const
Rectángulo de posición de la sombra.

Devuelve: Rectángulo de posición de la sombra.

void Bola::RestaurarPosicion ()
Restaurar a zona sin colisiones.
Restaura la posición de la bola a una zona libre de colisiones.

std::string Bola::RutaImagen (const char * *imagen*) `[private]`
Genera ruta a imagen.
Genera una ruta desde el directorio de imágenes de la bola.

Parámetros: ***imagen*** Nombre de la imagen.

Devuelve: Ruta a la imagen.

std::string Bola::RutaSonido (const char * *sonido*) `[private]`
Genera ruta a sonido.
Genera una ruta a un sonido desde el directorio de efectos de sonido.

Parámetros: ***imagen*** Nombre del sonido.

Devuelve: Ruta al sonido.

6.2.2.5. Documentación de los datos miembro

Sonido Bola::agua_ `[private]`
Sonido que genera la bola al caer al agua.

int Bola::altura_minima_ = 1 `[static, private]`
Altura mínima que puede tener la bola sobre su sombra.

Sonido Bola::arboles_ `[private]`
Sonido que genera la bola al caer a los árboles.

SDL_Surface* Bola::bola_ `[private]`
Imagen de la bola.

Sonido Bola::bunker_ `[private]`
Sonido que genera la bola al caer al bunker.

Datos::Bola Bola::desplazamiento_ `[private]`
Datos necesarios para restaurar la bola a una zona libre de colisión.

int Bola::estado_ `[private]`
Estado actual de la bola.

Sonido Bola::golpe_ `[private]`
Sonido que genera la bola al ser golpeada por un palo.

Sonido Bola::hoyo_ `[private]`
Sonido que genera la bola al entrar en el hoyo.

Rejilla∗ Bola::inactivas_ `[private]`
Imágenes de la bola cuando está inactiva.

Pantalla∗ Bola::pantalla_ `[private]`
Pantalla del juego.

Partida∗ Bola::partida_ `[private]`
Partida que está actualmente en juego.

SDL_Rect Bola::rbola_ `[private]`
Rectángulo de posición de la bola.

Sonido Bola::rebote_ `[private]`
Sonido que genera la bola al golpear en el campo.

SDL_Rect Bola::rsombra_ `[private]`
Rectángulo de posición de la sombra de la bola.

SDL_Surface ∗ Bola::sombra_ `[private]`
Imagen de la sombra de la bola.

int Bola::x_bola_ `[private]`

Posición inicial de la bola en el eje de abscisas.

int Bola::x_sombra_ `[private]`

Posición inicial la sombra de la bola en el eje de abscisas.

int Bola::y_bola_ `[private]`

Posición inicial de la bola en el eje de ordenadas.

int Bola::y_sombra_ `[private]`

Posición inicial de la sombra de la bola en el eje de ordenadas.

6.2.3. Clase Cámara

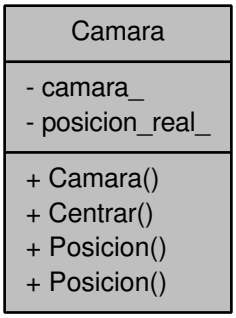


Figura 6.3: Detalle de la clase Cámara en UML

6.2.3.1. Descripción

Cámara de juego.

La clase cámara se encarga de centrar en la pantalla la posición del campo en el que se encuentra la bola de golf.

Cuando la bola de golf se está desplazando, la cámara la acompaña en su recorrido, simulando de esta forma la extensión del campo golf.

6.2.3.2. Documentación del constructor y destructor

Camara::Camara (int *ancho*, int *alto*)

Constructor.

Constructor de la cámara.

Establece el ancho y alto de la cámara, así como sus coordenadas.

Parámetros:

ancho Ancho de enfoque que tendrá la cámara.

alto Alto de enfoque que tendrá la cámara.

6.2.3.3. Documentación de las funciones miembro

void Camara::Centrar (const SDL_Rect & *objetivo*, const SDL_Rect & *superficie*)

Centra la cámara en un objetivo.

Centra la cámara en la posición indicada.

Parámetros:

objetivo Posición del objeto que tiene que enfocar la cámara.

superficie Superficie en la que se encuentra el objetivo.

SDL_Rect & Camara::Posicion (const SDL_Rect & *posicion*)

Posición relativa de un objeto.

Devuelve la posición relativa de un objeto con respecto a la cámara.

Devuelve: Posición del objeto con respecto a la cámara.

const SDL_Rect & Camara::Posicion () const

Posición de la cámara.

Devuelve: Posición en la que se encuentra la cámara.

6.2.3.4. Documentación de los datos miembro

SDL_Rect Camara::camara_ `[private]`

Rectángulo de posición de la cámara.

SDL_Rect Camara::posicion_real_ `[private]`

Rectángulo de posición para la posición real del objeto.

6.2.4. Clase Campo

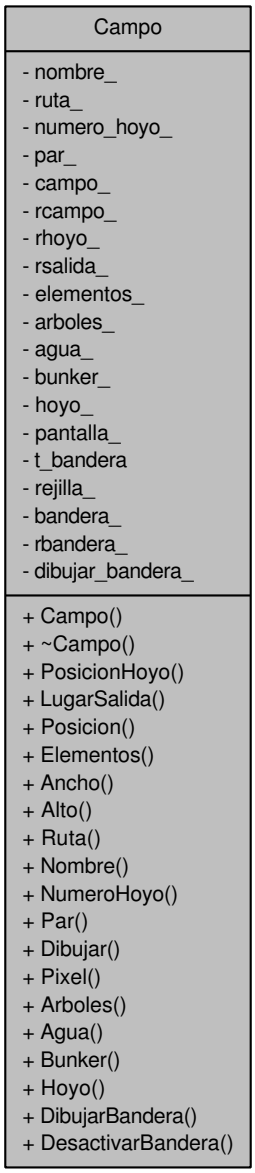


Figura 6.4: Detalle de la clase Campo en UML

6.2.4.1. Descripción

Campo de golf.

El campo de golf es la superficie en la que se desarrollará la partida.

Está compuesto por diferentes elementos con los que puede colisionar la bola de golf.

Todo campo de golf tiene una posición de salida para los jugadores y un hoyo en el que habrá que colar la bola.

6.2.4.2. Documentación de las enumeraciones miembro de la clase

enum Campo::ElementosCampo

Elementos de colisión del campo.

Elementos que componen el campo de golf con los que puede colisionar la bola.

Valores de la enumeración:

ARBOLES Árboles presentes en el terreno de juego.

AGUA Zonas de agua.

BUNKER Zonas de bunker

HOYO Hoyo del campo.

TODOS Todos los elementos.

FDL Fuera de límites.

6.2.4.3. Documentación del constructor y destructor

Campo::Campo (std::string *nombre*, std::string *hoyo*, std::string *par*, SDL_Rect * *posicion_salida*, SDL_Rect * *posicion_hoyo*, std::string *colores*, std::string *ruta*, Pantalla * *pantalla*)

Constructor.

Establece todos los atributos relacionados con el cambo (nombre, número de hoyo, par del campo...) y carga los colores de colisión de los diferentes elementos del campo.

De igual forma, se crea la animación de la bandera.

Parámetros:

nombre Nombre del campo.

hoyo Número de hoyo del campo.

par Par del hoyo.

posicion_salida Posición de salida de los jugadores.

posicion_hoyo Posición en la que se encuentra el hoyo.

colores Nombre del fichero XML que contiene los colores de colisión.

ruta Ruta a las imágenes del campo.

pantalla Pantalla del juego.

Campo::∼Campo ()

Destructor.

Destruye la imagen del campo, la rejilla de imágenes de la bandera y la animación de la bandera.

6.2.4.4. Documentación de las funciones miembro

const std::vector< Uint32 > & Campo::Agua () const

Agua.

Devuelve: Colores del agua.

int Campo::Alto () const

Alto del terreno.

Devuelve: Alto de la superficie del campo.

int Campo::Ancho () const

Ancho del terreno.

Devuelve: Ancho de la superficie del campo.

const std::vector< Uint32 > & Campo::Arboles () const

Árboles.

Devuelve: Colores de los árboles.

const std::vector< Uint32 > & Campo::Bunker () const

Bunker.

Devuelve: Colores del bunker.

void Campo::DesactivarBandera ()

Quita la bandera.

Desactiva la bandera para evitar que se siga dibujando.

void Campo::Dibujar ()
Dibujar campo.
Dibuja el campo en la pantalla.

void Campo::DibujarBandera ()
Dibuja la bandera.
Dibuja la animación de la bandera.

const std::map< int, std::vector< Uint32 > > & Campo::Elementos () const
Elementos.

Devuelve: Elementos que componen el campo y con los que puede colisionar la bola.

const std::vector< Uint32 > & Campo::Hoyo () const
Hoyo.

Devuelve: Colores del hoyo.

const SDL_Rect & Campo::LugarSalida () const
Posición de salida.

Devuelve: Posición de salida para los jugadores.

std::string Campo::Nombre () const
Nombre del campo.

Devuelve: Nombre del campo.

std::string Campo::NumeroHoyo () const
Número de hoyo.

Devuelve: Número de hoyo del campo.

int Campo::Par () const
Par del hoyo.

Devuelve: Par del hoyo.

Uint32 Campo::Pixel (int *x*, int *y*) const

Obtener Píxel.

Obtiene el color de un determinado píxel.

Parámetros:

- ***x*** Coordenada x del píxel.
- ***y*** Coordenada y del píxel.

const SDL_Rect & Campo::Posicion () const

Posición.

Devuelve: Posición del campo.

const SDL_Rect & Campo::PosicionHoyo () const

Posición del hoyo.

Devuelve: Posición del hoyo del campo.

std::string Campo::Ruta () const

Ruta a las imágenes.

Devuelve: Ruta a las imágenes del campo.

6.2.4.5. Documentación de los datos miembro

std::vector<Uint32> Campo::agua_ `[private]`

Colores del agua.

std::vector<Uint32> Campo::arboles_ `[private]`

Colores de los árboles.

Animacion* Campo::bandera_ `[private]`

Animación de la bandera.

std::vector<Uint32> Campo::bunker_ `[private]`

Colores del bunker.

SDL_Surface* Campo::campo_ `[private]`
Imagen de la superficie del campo.

bool Campo::dibujar_bandera_ `[private]`
Variable de control para el dibujo de la bandera.

std::map< int, std::vector<Uint32> > Campo::elementos_ `[private]`
Elementos del campo de golf.

std::vector<Uint32> Campo::hoyo_ `[private]`
Colores del hoyo.

std::string Campo::nombre_ `[private]`
Nombre del campo.

std::string Campo::numero_hoyo_ `[private]`
Número de hoyo.

Pantalla* Campo::pantalla_ `[private]`
Pantalla del juego.

std::string Campo::par_ `[private]`
Par del campo.

SDL_Rect Campo::rbandera_ `[private]`
Posición de la bandera.

SDL_Rect Campo::rcampo_ `[private]`
Rectángulo de posición de la imagen del campo.

Rejilla* Campo::rejilla_ `[private]`
Rejilla de imágenes de la bandera.

SDL_Rect Campo::rhoyo_ `[private]`
Rectángulo de posición del hoyo.

SDL_Rect Campo::rsalida_ `[private]`

Rectángulo de posición de la salida de los jugadores.

std::string Campo::ruta_ `[private]`

Ruta a las imágenes del campo.

Uint32 Campo::t_bandera `[private]`

Control de tiempo para la animación de la bandera.

6.2.5. Clase Cursor

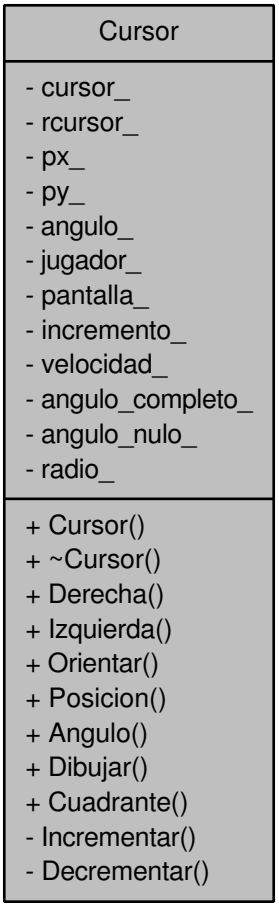


Figura 6.5: Detalle de la clase Cursor en UML

6.2.5.1. Descripción

Cursor de dirección.

El cursor de dirección permite al jugador visualizar hacia dónde está apuntando.

El jugador podrá establecer la dirección que desee girando el cursor hacia la derecha o hacia la izquierda.

La dirección que indique el cursor será la que tome la bola de golf cuando sea golpeada.

6.2.5.2. Documentación del constructor y destructor

Cursor::Cursor (Jugador * *jugador*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con el jugador y la pantalla y establece el tamaño y las coordenadas iniciales del cursor.

Parámetros:

jugador Jugador al que pertenece el cursor.

pantalla Pantalla donde se dibujará el cursor.

Cursor::~Cursor ()

Destructor.

Destruye la imagen del cursor.

6.2.5.3. Documentación de las funciones miembro

float Cursor::Angulo () const

Ángulo del cursor.

Devuelve: Ángulo que forma el cursor con la bola.

int Cursor::Cuadrante () const

Cuadrante.

Devuelve: Cuadrante en el que se encuentra el cursor.

void Cursor::Decrementar () `[private]`

Decrementamos el ángulo del cursor.

void Cursor::Derecha ()

Gira a la derecha.

Gira el cursor a la derecha.

void Cursor::Dibujar () const
Dibuja el cursor.
Dibuja el cursor en la pantalla.

void Cursor::Incrementar () `[private]`
Incrementamos el ángulo del cursor.

void Cursor::Izquierda ()
Gira a la izquierda.
Gira el cursor a la izquierda.

void Cursor::Orientar ()
Orienta al hoyo.
Orienta el cursor en dirección al hoyo del campo.

const SDL_Rect & Cursor::Posicion () const
Posición del cursor.

Devuelve: Posición del cursor.

6.2.5.4. Documentación de los datos miembro

float Cursor::angulo_ `[private]`
Incremento del ángulo del cursor.

float Cursor::angulo_completo_ = M_PI * 2 `[static, private]`
Ángulo completo.

float Cursor::angulo_nulo_ = 0 `[static, private]`
Ángulo nulo.

SDL_Surface* Cursor::cursor_ `[private]`
Imagen del cursor.

float Cursor::incremento_ = M_PI / 180 `[static, private]`
Incremento del ángulo del cursor.

Jugador* Cursor::jugador_ `[private]`

Jugador con el que se relaciona el cursor.

Pantalla* Cursor::pantalla_ `[private]`

Pantalla del juego.

int Cursor::px_ `[private]`

Coordenada x del cursor.

int Cursor::py_ `[private]`

Coordenada y del cursor.

float Cursor::radio_ = 70 `[static, private]`

Radio del cursor.

SDL_Rect Cursor::rcursor_ `[private]`

Posición del cursor.

float Cursor::velocidad_ = 2 `[static, private]`

Velocidad de giro del cursor.

6.2.6. Clase Humano

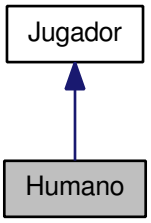


Figura 6.6: Diagrama de herencia de la clase Humano

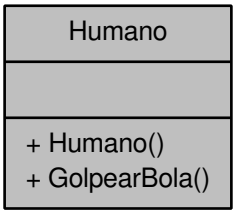


Figura 6.7: Detalle de la clase Humano en UML

6.2.6.1. Descripción

Jugador controlado por una persona.

Esta clase se encarga de la implementación de un jugador controlado por una persona.

Todos los jugadores tienen un nombre, un color y pueden pertenecer a un equipo cuando se está jugando una partida por parejas.

De la misma forma, los jugadores de un mismo equipo podrán compartir la bola de golf. A parte de esto, el jugador cuenta con una lista de palos de golf que podrá utilizar durante la partida.

6.2.6.2. Documentación del constructor y destructor

Humano::Humano (std::string *nombre*, int *color*, Bola & *bola*, Partida * *partida*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con la partida y la pantalla.

Establece el nombre y el color del jugador. Carga las imágenes y crea la lista de palos. Genera las animaciones del jugador.

Parámetros:

nombre Nombre del jugador.

color Color del jugador.

bola Bola del jugador.

partida Partida que se está desarrollando.

pantalla Pantalla donde se dibujará la bola.

6.2.6.3. Documentación de las funciones miembro

int Humano::GolpearBola () `[virtual]`

Golpea la bola.

El jugador realiza un lanzamiento de la bola golpeándola con el palo.

Devuelve: **0** cuando el lanzamiento ha finalizado.

6.2.7. Clase Indicador

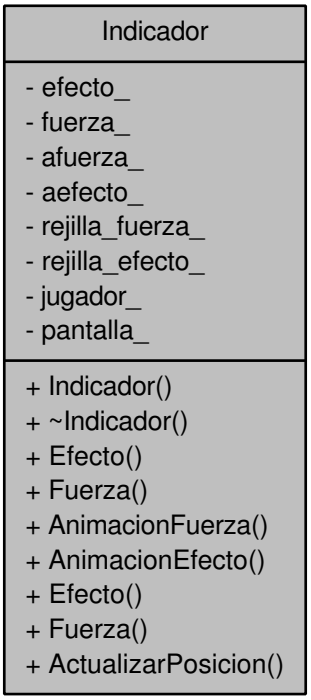


Figura 6.8: Detalle de la clase Indicador en UML

6.2.7.1. Descripción

Indicador de fuerza y efecto.

La clase indicador se encarga de mostrar al usuario los niveles de fuerza y efecto que se aplicarán al lanzamiento.

Para ello, se mostrarán sendas animaciones con las que el usuario podrá interactuar para determinar los niveles de fuerza y efectos deseados con los que se realizará el lanzamiento de la bola de golf.

6.2.7.2. Documentación del constructor y destructor

Indicador::Indicador (Jugador $*$ *jugador*, Pantalla $*$ *pantalla*)

Constructor.

Realiza las asociaciones con el jugador y la pantalla.

Genera las animaciones para los indicadores de fuerza y efecto.

Parámetros:

jugador Jugador al que pertenece el indicador.

pantalla Pantalla donde se dibujará el indicador.

Indicador::~Indicador ()

Destructor.

Destruye las animaciones y las rejillas de imágenes de los indicadores de fuerza y efecto.

6.2.7.3. Documentación de las funciones miembro

void Indicador::ActualizarPosicion ()

Actualiza la posición del indicador.

Actualiza la posición de los indicadores en función del cuadro del jugador que esté seleccionado.

Animacion & Indicador::AnimacionEfecto ()

Animación del indicador de efecto.

Devuelve la animación del indicador de efecto.

Devuelve: Animación de la barra de efecto.

Animacion & Indicador::AnimacionFuerza ()

Animación del indicador de fuerza.

Devuelve la animación del indicador de fuerza.

Devuelve: Animación de la barra de fuerza.

int Indicador::Efecto () const

Efecto del lanzamiento.

Devuelve: Valor del efecto aplicado a la bola.

void Indicador::Efecto (int *efecto*)

Efecto de tiro.

Establece el valor de efecto que se ha aplicado a la bola en el lanzamiento.

Parámetros: ***efecto*** Valor del efecto aplicado.

int Indicador::Fuerza () const

Fuerza del lanzamiento.

Devuelve: Valor de la fuerza aplicada a la bola.

void Indicador::Fuerza (int *fuerza*)

Establece la fuerza de lanzamiento.

Establece el valor de la fuerza que se ha aplicado a la bola en el lanzamiento.

Parámetros: ***fuerza*** Valor de la fuerza aplicada.

6.2.7.4. Documentación de los datos miembro

Animacion* Indicador::aefecto_ `[private]`

Animación de la barra de efecto.

Animacion* Indicador::afuerza_ `[private]`

Animación de la barra de fuerza.

int Indicador::efecto_ `[private]`

Efecto aplicado a la bola.

int Indicador::fuerza_ `[private]`

Fuerza aplicada a la bola.

Jugador* Indicador::jugador_ `[private]`

Jugador con el que se relaciona el indicador.

Pantalla* Indicador::pantalla_ `[private]`

Pantalla del juego.

Rejilla* Indicador::rejilla_efecto_ `[private]`

Rejilla de la barra de efecto.

Rejilla* Indicador::rejilla_fuerza_ `[private]`

Rejilla de la barra de fuerza.

6.2.8. Clase Juego

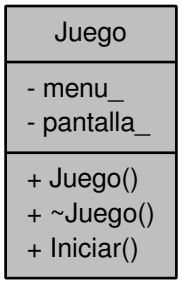


Figura 6.9: Detalle de la clase Juego en UML

6.2.8.1. Descripción

Clase principal.

Esta es la clase principal del juego.

En ella, se inicializan los subsistemas de imagen, sonido y fuentes y se crea la pantalla en la que se dibujarán todos los elementos gráficos del juego.

6.2.8.2. Documentación del constructor y destructor

Juego::Juego ()

Constructor.

Inicializa SDL, los subsistemas de sonido, imagen y fuentes. Crea el menú y lo muestra.

Juego::∼Juego ()

Destructor.

Destruye el menú.

6.2.8.3. Documentación de las funciones miembro

void Juego::Iniciar ()

Inicia el juego.

Inicia el juego invocando al menú principal.

6.2.8.4. Documentación de los datos miembro

Menu∗ Juego::menu_ `[private]`

Menú del juego.

Pantalla Juego::pantalla_ `[private]`

Pantalla del juego.

6.2.9. Clase Jugador

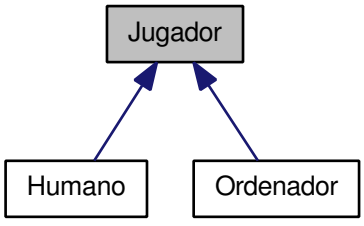


Figura 6.10: Diagrama de herencia de la clase Jugador

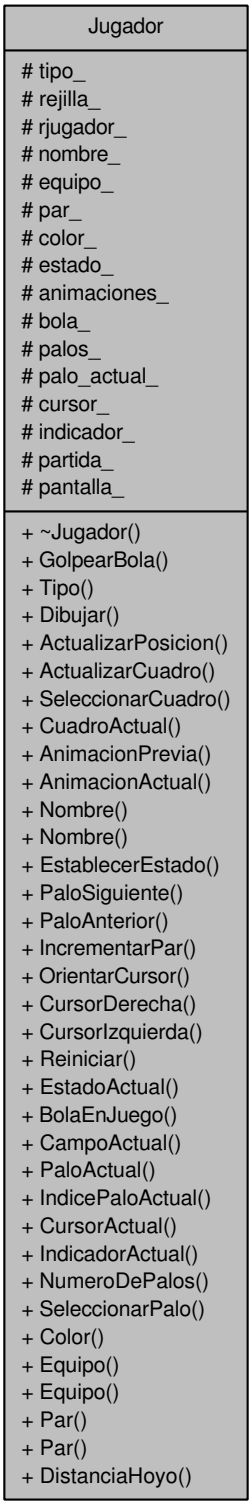


Figura 6.11: Detalle de la clase Jugador en UML

6.2.9.1. Descripción

Clase virtual.

Los jugadores pueden ser personas o estar controlados por el ordenador.

Esta clase engloba todos los métodos necesarios para la implementación de los jugadores.

Todos los jugadores tienen un nombre, un color y pueden pertenecer a un equipo cuando se está jugando una partida por parejas. De la misma forma, los jugadores de un mismo equipo podrán compartir la bola de golf. A parte de esto, el jugador cuenta con una lista de palos de golf que podrá utilizar durante la partida.

6.2.9.2. Documentación de las enumeraciones miembro de la clase

enum Jugador::Estados

Estados del jugador.

Estados en los que puede encontrarse el jugador.

Valores de la enumeración:

ACTIVO El jugador está jugando.

INACTIVO El jugador está a la espera de entrar en juego.

FINALIZADO El jugador ha efectuado hoyo.

enum Jugador::Tipos

Tipos de jugadores.

Tipos de jugadores a los que puede pertenecer un jugador.

Valores de la enumeración:

HUMANO El jugador es una persona.

IA_BAJA El jugador es el ordenador y la inteligencia artificial es baja.

IA_MEDIA El jugador es el ordenador y la inteligencia artificial es media.

IA_ALTA El jugador es el ordenador y la inteligencia artificial es alta.

6.2.9.3. Documentación del constructor y destructor

Jugador::∼Jugador ()

Destructor.

Destruye la rejilla, el cursor, el indicador de fuerza y efecto y todas las animaciones.

6.2.9.4. Documentación de las funciones miembro

void Jugador::ActualizarCuadro ()

Actualiza la imagen del jugador dependiendo de donde apunte el cursor.

Actualiza el cuadro de la rejilla en función de la posición del cursor.

void Jugador::ActualizarPosicion ()

Actualiza la posición dependiendo de la imagen de la rejilla.

Actualiza la posición del jugador en función del cuadro de la rejilla que esté seleccionado.

Animacion & Jugador::AnimacionActual ()

Animación del jugador.

Devuelve la animación del jugador.

Devuelve: Animación actual del jugador en función del cuadro de la rejilla.

void Jugador::AnimacionPrevia ()

Animación previa al lanzamiento.

Lleva a cabo la secuencia de animación del jugador que se corresponde con el intervalo de tiempo entre que el jugador prepara el tiro hasta que golpea la bola.

Bola & Jugador::BolaEnJuego () const

Bola del jugador.

Devuelve: Bola del jugador.

Campo & Jugador::CampoActual () const

Campo en el que se está jugando.

Devuelve: Campo en el que se desarrolla la partida actual.

int Jugador::Color () const

Color del jugador.

Devuelve: Color del jugador.

int Jugador::CuadroActual () const

Cuadro del jugador seleccionado.

Devuelve: Cuadro actual seleccionado en la rejilla.

Cursor & Jugador::CursorActual ()

Cursor del jugador.

Devuelve: Cursor del jugador.

void Jugador::CursorDerecha ()

Gira el cursor del jugador a la derecha.

void Jugador::CursorIzquierda ()

Gira el cursor del jugador a la izquierda.

void Jugador::Dibujar () const

Dibuja el jugador.

Dibuja el cuadro del jugador en la pantalla.

int Jugador::DistanciaHoyo () const

Distancia al hoyo en línea recta.

Devuelve: Distancia al hoyo del campo a la que se encuentra el jugador.

int Jugador::Equipo () const

Número de equipo del jugador.

Devuelve: Número de equipo del jugador.

void Jugador::Equipo (int *equipo*)

Establece el equipo al que pertenece el jugador.

Parámetros: ***equipo*** Número de equipo del jugador.

void Jugador::EstablecerEstado (int *estado*)

Establece el estado del jugador.

Parámetros: ***estado*** Estado del jugador.

int Jugador::EstadoActual () const

Estado del jugador.

Devuelve: Estado del jugador.

virtual int Jugador::GolpearBola () `[pure virtual]`

Golpeo de la bola.

El jugador golpea la bola.

Devuelve: 0 cuando el lanzamiento ha finalizado.

Implementado en **Humano**, y **Ordenador**.

void Jugador::IncrementarPar ()

Incrementa en una unidad el par del jugador.

Indicador & Jugador::IndicadorActual ()

Indicador de fuerza y efecto del jugador.

Devuelve: Indicador del jugador.

int Jugador::IndicePaloActual () const

Palo seleccionado por el jugador.

Devuelve: Palo que el jugador tiene seleccionado actualmente.

std::string Jugador::Nombre ()

Nombre del jugador.

Devuelve: Nombre del jugador.

void Jugador::Nombre (std::string *nombre*)
Establece el nombre del jugador.

Parámetros: ***nombre*** Nombre del jugador.

int Jugador::NumeroDePalos () const
Número de palos disponibles.

Devuelve: Número de palos disponibles.

void Jugador::OrientarCursor ()
Orienta el cursor del jugador al hoyo del campo.

const Palo & Jugador::PaloActual () const
Palo seleccionado por el jugador.

Devuelve: Palo que el jugador tiene seleccionado actualmente.

void Jugador::PaloAnterior ()
Selecciona el palo anterior al actual.

void Jugador::PaloSiguiente ()
Selecciona el palo siguiente al actual.

int Jugador::Par () const
Par del jugador.

Devuelve: Par del jugador.

void Jugador::Par (int *par*)
Establece el par del jugador.

Parámetros: ***Par*** del jugador.

void Jugador::Reiniciar ()
Establece el par del jugador a 0 y su estado a inactivo.

void Jugador::SeleccionarCuadro (int *cuadro*)

Selecciona un cuadro de la rejilla.

Selecciona un determinado cuadro en la rejilla.

Parámetros: ***cuadro*** Número de cuadro a seleccionar.

void Jugador::SeleccionarPalo (std::string *palo*)

Selecciona el palo indicado.

Parámetros: ***Nombre*** del palo que queremos seleccionar.

int Jugador::Tipo () const

Tipo de jugador.

Devuelve: Tipo de jugador.

6.2.9.5. Documentación de los datos miembro

std::vector<Animacion *> Jugador::animaciones_ `[protected]`

Animaciones del jugador.

Bola* Jugador::bola_ `[protected]`

Bola asociada al jugador.

int Jugador::color_ `[protected]`

Color del jugador.

Cursor* Jugador::cursor_ `[protected]`

Cursor del jugador.

int Jugador::equipo_ `[protected]`

Equipo al que pertenece el jugador.

int Jugador::estado_ `[protected]`

Estado actual del jugador.

Indicador* Jugador::indicador_ `[protected]`

Indicador de fuerza y efecto del jugador.

std::string Jugador::nombre_ `[protected]`

Nombre del jugador.

unsigned Jugador::palo_actual_ `[protected]`

Palo seleccionado actualmente.

std::deque<Palo> Jugador::palos_ `[protected]`

Lista de palos del jugador.

Pantalla* Jugador::pantalla_ `[protected]`

Pantalla del juego.

int Jugador::par_ `[protected]`

Par del jugador.

Partida* Jugador::partida_ `[protected]`

Partida que está actualmente en juego.

Rejilla* Jugador::rejilla_ `[protected]`

Rejilla de imágenes del jugador.

SDL_Rect Jugador::rjugador_ `[protected]`

Posición de la imagen del jugador.

int Jugador::tipo_ `[protected]`

Tipo de jugador.

6.2.10. Clase Matchplay

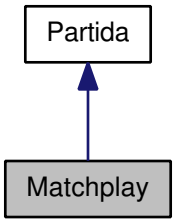


Figura 6.12: Diagrama de herencia de la clase Matchplay

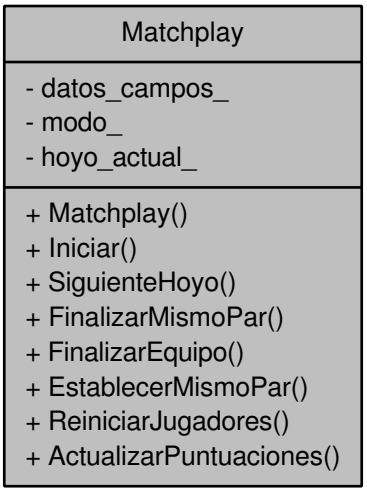


Figura 6.13: Detalle de la clase Matchplay en UML

6.2.10.1. Descripción

Modo de juego Matchplay.

Implementación del modo de juego Matchplay.

En el modo Matchplay, dos o más jugadores se enfrentan por parejas en todos los hoyos de un campo.

El equipo o el jugador que gane un hoyo sumará un punto. El equipo o el jugador que pierda el hoyo no sumará nada. Si los equipos o los jugadores empatan un hoyo, tampoco sumarán puntos.

Los miembros de un mismo equipo podrán compartir la bola. Si se comparte la bola, los jugadores también compartirán el par.

Ganará la partida el equipo o el jugador que más puntos haya conseguido.

6.2.10.2. Documentación de las enumeraciones miembro de la clase

enum Matchplay::Modos

Modos de juego para Matchplay.

Valores de la enumeración:

BOLAS_INDEPENDIENTES Cada jugador tiene su propia bola.

BOLAS_COMPARTIDAS Los jugadores del mismo equipo comparten la bola.

6.2.10.3. Documentación del constructor y destructor

Matchplay::Matchplay (std::vector< Datos::Jugador > & *jugadores*, std::vector< Datos::Campo > & *campos*, int *modo*, Menu * *menu*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con el menú y la pantalla.

Crea y asigna las bolas de golf y los jugadores de los diferentes equipos.

Inicializa las puntuaciones a 0 y carga el primer campo.

Coloca las Bolas sobre el campo, actualiza la lista de bolas inactivas y activa el primer jugador.

Crea las animaciones de las colisiones de la bola con los elementos del campo.

Parámetros:

jugadores Datos de los jugadores que participan en la partida.

campos Datos de los campos en los que se desarrollará la partida.

modo Modo de juego para Matchplay.

menu Menú del juego.

pantalla Pantalla del juego.

6.2.10.4. Documentación de las funciones miembro

void Matchplay::ActualizarPuntuaciones ()

Actualiza las puntuaciones.

Actualiza las puntuaciones de los equipos.

El equipo que posea el menor par sumará un punto. Si los equipos tienen el mismo par no suman nada.

void Matchplay::EstablecerMismoPar (int *par*, int *equipo*)

Establece el mismo par a un equipo.

Establece el mismo par a los jugadores que estén en el mismo equipo.

Parámetros:

par Par que posee el jugador activo.

equipo Equipo al que pertenece el jugador activo.

void Matchplay::FinalizarEquipo (int *equipo*)

Finaliza a todos los jugadores de un equipo.

Parámetros: ***equipo*** Equipo a finalizar.

void Matchplay::FinalizarMismoPar (int *par*, int *equipo*)

Finaliza los jugadores del mismo par.

Finaliza los jugadores del mismo equipo que posean el mismo par que el jugador activo o un par menor en una unidad, puesto que no podrán mejorar la puntuación del equipo.

Parámetros:

par Par que posee el jugador activo.

equipo Equipo al que pertenece el jugador activo.

int Matchplay::Iniciar () `[virtual]`

Inicia la partida.

Devuelve:

- **0** hemos salido de la partida desde el menú de pausa.
- **-1** hemos cerrado la ventana del juego.

Implementa **Partida**.

void Matchplay::ReiniciarJugadores ()

Reinicia todos los jugadores de la partida.

bool Matchplay::SiguienteHoyo ()

Pasa al siguiente hoyo.

Pasa al siguiente hoyo, destruyendo el actual y modificando el campo activo.

Devuelve:

- **Verdadero** en caso de haber más hoyos.
- **Falso** en caso de no haber más hoyos.

6.2.10.5. Documentación de los datos miembro

std::vector<Datos::Campo> Matchplay::datos_campos_ `[private]`

Datos de los campos de la partida.

unsigned int Matchplay::hoyo_actual_ `[private]`

Hoyo en el que se está jugando.

int Matchplay::modo_ `[private]`

Modo de Matchplay.

6.2.11. Clase Ordenador

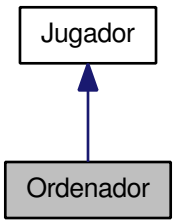


Figura 6.14: Diagrama de herencia de la clase Ordenador

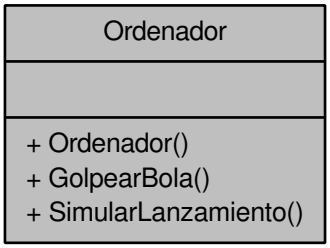


Figura 6.15: Detalle de la clase Ordenador en UML

6.2.11.1. Descripción

Jugador controlado por el ordenador.

Esta clase se encarga de la implementación de un jugador controlado por el ordenador, es decir, en ella se desarrolla la inteligencia artificial del juego.

Todos los jugadores tienen un nombre, un color y pueden pertenecer a un equipo cuando se está jugando una partida por parejas. De la misma forma, los jugadores de un mismo equipo podrán compartir la bola de golf. A parte de esto, el jugador cuenta con una lista de palos de golf que podrá utilizar durante la partida.

6.2.11.2. Documentación del constructor y destructor

Ordenador::Ordenador (std::string *nombre*, int *color*, int *tipo*, Bola & *bola*, Partida * *partida*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con la partida y la pantalla.

Establece el nombre y el color del jugador. Carga las imágenes y crea la lista de palos. Genera las animaciones del jugador.

Parámetros:

nombre Nombre del jugador.

color Color del jugador.

tipo Tipo de jugador.

bola Bola del jugador.

partida Partida que se está desarrollando.

pantalla Pantalla donde se dibujará la bola.

6.2.11.3. Documentación de las funciones miembro

int Ordenador::GolpearBola () `[virtual]`

Golpea la bola.

El ordenador realiza el lanzamiento de la bola.

Busca el mejor lanzamiento en función del nivel de inteligencia.

Devuelve: **0** cuando el lanzamiento ha finalizado.

Implementa **Jugador**.

int Ordenador::SimularLanzamiento (int *x*, int *y*, const Palo & *palo*)

Simula el lanzamiento de la bola.

Devuelve:

- **-1** Si se ha producido colisión en el lanzamiento.
- **Distancia** a la que estaría la bola del hoyo del campo, en otro caso.

6.2.12. Clase Palo

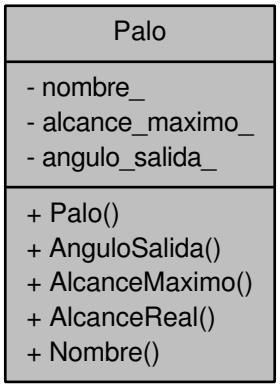


Figura 6.16: Detalle de la clase Palo en UML

6.2.12.1. Descripción

Palo de golf.

El palo de golf es uno de los elementos básicos del juego. Con él se podrá llevar a cabo el lanzamiento de la bola de golf.

Cada palo de golf viene determinado por unas características propias. Todos poseen un nombre, un alcance máximo y un ángulo de tiro. Cuanto mayor sea dicho el ángulo de tiro, más altura podrá alcanzar la bola pero menos alcance tendrá el lanzamiento.

6.2.12.2. Documentación del constructor y destructor

Palo::Palo (std::string *nombre*, int *alcance_maximo*, int *angulo_salida*)

Constructor.

Establece el nombre, el alcance y el ángulo de salida del palo.

Parámetros:

nombre Nombre del palo de golf.

alcance_maximo Alcance máximo (en metros) del palo de golf.

angulo_salida Ángulo de salida de la bola al ser golpeada por el palo.

6.2.12.3. Documentación de las funciones miembro

int Palo::AlcanceMaximo () const

Alcance máximo del palo de golf convertido a la proporción del juego.

Devuelve: Alcance máximo del palo.

int Palo::AlcanceReal () const

Alcance máximo (en metros) del palo de golf.

Devuelve: Alcance real del palo.

int Palo::AnguloSalida () const

Ángulo de salida de la bola al ser golpeada por el palo.

Devuelve: Ángulo de salida.

std::string Palo::Nombre () const

Nombre del palo de golf.

Devuelve: Nombre del palo.

6.2.12.4. Documentación de los datos miembro

int Palo::alcance_maximo_ `[private]`

Alcane máximo real.

int Palo::angulo_salida_ `[private]`

Ángulo de salida que aplica a la bola.

std::string Palo::nombre_ `[private]`

Nombre del palo.

6.2.13. Clase Partida

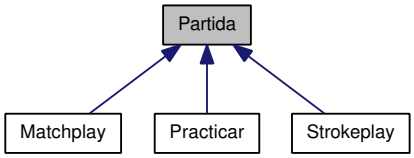


Figura 6.17: Diagrama de herencia de la clase Partida

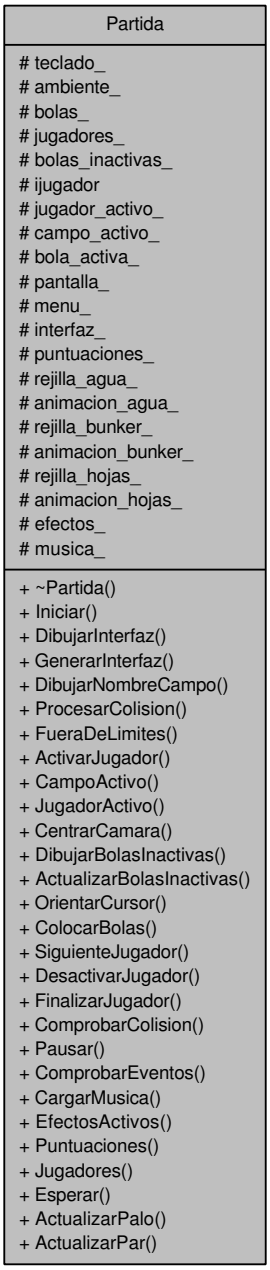


Figura 6.18: Detalle de la clase Partida en UML

6.2.13.1. Descripción

Clase virtual.

La clase partida engloba todos las funciones compartidas por los diferentes modos de juego.

Desde esta clase se podrá controlar y acceder todos los elementos que participan en una partida.

La clase partida también se encarga de detectar y procesar las colisiones de la bola de golf con los elementos del campo.

6.2.13.2. Documentación de las enumeraciones miembro de la clase

enum Partida::ModosComprobacion

Modos de comprobación de colisión.

Enumeración de los modos de comprobación de colisión.

Valores de la enumeración:

CENTRO Comprueba la colisión en el píxel del centro de la bola.

COMPLETO Comprueba la colisión en todos los píxeles de la bola.

RESTAURACION Comprueba la colisión en todos los píxeles de la bola y en una distancia determinada a su alrededor.

6.2.13.3. Documentación del constructor y destructor

Partida::~Partida () `[virtual]`

Destructor virtual de partida.

Destruye las bolas, los jugadores, el campo, las animaciones y la interfaz.

6.2.13.4. Documentación de las funciones miembro

void Partida::ActivarJugador (Jugador * *jugador*)

Activa un determinado jugador.

Parámetros: ***jugador*** Jugador que se quiere activar.

void Partida::ActualizarBolasInactivas ()

Actualiza la lista de bolas inactivas.

void Partida::ActualizarPalo ()

Actualiza el palo de la interfaz.

void Partida::ActualizarPar ()

Actualiza el par del jugador en la interfaz.

Campo & Partida::CampoActivo ()

Devuelve el campo activo.

Devuelve: Campo en el que se está desarrollando la partida actualmente.

void Partida::CargarMusica ()

Carga la música ambiente de la partida.

Se elige una pista aleatoria entre las disponibles.

void Partida::CentrarCamara ()

Centra la cámara en la bola.

void Partida::ColocarBolas ()

Coloca las bolas de la partida partiendo de la posición inicial.

bool Partida::ComprobarColision (int *modo*, int *elemento*, SDL_Rect * *bola*, SDL_Rect * *sombra*)

Comprueba la colisión de la bola con los elementos del campo.

Parámetros:

modo Modo de comprobación de colisión.

elemento Elemento del campo a comprobar.

bola Posición de la bola con la que se llevará a cabo la comprobación.

sombra Posición de la sombra de la bola.

Nota: Si *bola* o *sombra* son nulos, se tomará la posición de la bola del jugador activo para la comprobación de la colisión.

int Partida::ComprobarEventos ()

Comprueba los eventos del usuario.

Comprueba los eventos que genera el usuario mientras está jugando el ordenador y los procesa en caso de que se cierre la ventana o el juego sea pausado.

Devuelve:

- **-1** Si se sale de la partida.
- **1** Si se ha cerrado la ventana del juego.
- **0** Si se continua la partida.

void Partida::DesactivarJugador (Jugador * *jugador*)

Desactiva a un determinado jugador.

Parámetros: ***jugador*** Jugador que se quiere desactivar.

void Partida::DibujarBolasInactivas ()

Dibuja en la pantalla las bolas que esten inactivas.

void Partida::DibujarInterfaz ()

Dibuja la interfaz de la partida.

void Partida::DibujarNombreCampo ()

Muestra el nombre del campo.

Muestra un rótulo con el nombre del campo y el número de hoyo que se va a jugar.

bool Partida::EfectosActivos () const

Estado de los efectos de sonido.

Función observadora. Devuelve si los efectos de sonido están activados o no.

Devuelve:

- **Verdadero** Si los efectos de sonido están activados.
- **Falso** Si los efectos de sonido no están activados.

void Partida::Esperar (Uint32 *ms*, bool *bola*, bool *sombra*, bool *cursor*, bool *fuerza*, bool *efecto*, SDL_Surface * *texto*, SDL_Rect * *rtexto*)

Espera un determinado intervalo de tiempo.

Pone la partida en espera un determinado tiempo permitiendo que la animación de la bandera se siga dibujando.

También permite realizar la espera cuando se dibuja el rótulo del resultado de una colisión.

Cuando la partida está en espera el juego no responde a los eventos que pueda producir el usuario.

Parámetros:

ms Tiempo en milisegundos que se desea esperar.

bola Indica si se ha de dibujar la bola.

sombra Indica si se ha de dibujar la sombra.

cursor Indica si se ha de dibujar el cursor.

fuerza Indica si se ha de dibujar el indicador de fuerza.

efecto Indica si se ha de dibujar el indicador de efecto.

texto Imagen del texto que se desea dibujar.

rtexto Posición del texto.

void Partida::FinalizarJugador (Jugador * *jugador*)

Finaliza a un determinado jugador.

Parámetros: ***jugador*** Jugador que se quiere finalizar.

bool Partida::FueraDeLimites (SDL_Rect * *bola*, SDL_Rect * *sombra*)

Comprueba que la bola esté dentro de los límites del campo.

Parámetros:

bola Posición de la bola.

sombra Posición de la sombra de la bola.

Devuelve:

- **Verdadero** Si la bola se encuentra fuera de los límites de la pantalla.
- **Falso** Si la bola se encuentra dentro de los límites de la pantalla.

void Partida::GenerarInterfaz ()

Actualiza la interfaz.

Actualiza la interfaz de la partida con los nuevos valores.

virtual int Partida::Iniciar () `[pure virtual]`

Función virtual.

Inicia la partida.

Devuelve:

- **-1** Si hemos cerrado la ventana.
- **0** En otro caso.

Implementado en **Matchplay**, **Practicar** y **Strokeplay**.

Jugador & Partida::JugadorActivo ()

Devuelve el jugador activo.

Devuelve: Jugador que está activo en la partida actualmente.

const std::vector< Jugador * > & Partida::Jugadores () const

Jugadores de la partida.

Función observadora. Devuelve los jugadores que participan en la partida.

Devuelve: Jugadores que participan en la partida.

void Partida::OrientarCursor ()

Orienta el cursor al hoyo del campo.

int Partida::Pausar ()

Pausa la partida actual.

Devuelve:

- **-1** Si se sale de la partida.
- **1** Si se ha cerrado la ventana del juego.
- **0** Si se continua la partida.

void Partida::ProcesarColision (int *elemento*)

Procesa una determinada colisión.

Parámetros: ***elemento*** Elemento con el que se ha producido la colisión.

const std::vector< int > & Partida::Puntuaciones () const

Puntuaciones de los jugadores.

Función observadora. Devuelve las puntuaciones de los jugadores que participan en la partida.

Devuelve: Puntuaciones de los jugadores.

bool Partida::SiguienteJugador ()

Activa el siguiente jugador.

Cambia el puntero del jugador activo al siguiente jugador que esté en estado inactivo.

Devuelve:

- **Verdadero** Si se ha realizado el cambio puesto que existían jugadores inactivos.
- **Falso** Si no se ha producido el cambio dado que no existían jugadores inactivos.

6.2.13.5. Documentación de los datos miembro

Musica Partida::ambiente_ `[protected]`

Música de fondo.

Animacion* Partida::animacion_agua_ `[protected]`

Animación del agua.

Animacion* Partida::animacion_bunker_ `[protected]`

Animación del bunker.

Animacion* Partida::animacion_hojas_ `[protected]`

Animación de las hojas.

Bola* Partida::bola_activa_ `[protected]`

Bola del jugador activo.

std::vector<Bola*> Partida::bolas_ [protected]
Bolas de la partida.

std::vector<Bola*> Partida::bolas_inactivas_ [protected]
Bolas en estado inactivo.

Campo* Partida::campo_activo_ [protected]
Campo que está activo.

bool Partida::efectos_ [protected]
Variable de control de los efectos de sonido.

std::vector<Jugador*>::iterator Partida::ijugador [protected]
Iterador de jugadores.

Interfaz* Partida::interfaz_ [protected]
Interfaz de la partida.

Jugador* Partida::jugador_activo_ [protected]
Jugador que está activo.

std::vector<Jugador*> Partida::jugadores_ [protected]
Jugadores que participan en la partida.

Menu* Partida::menu_ [protected]
Menú del juego.

bool Partida::musica_ [protected]
Variable de control de la música del juego.

Pantalla* Partida::pantalla_ [protected]
Pantalla del juego.

std::vector<int> Partida::puntuaciones_ [protected]
Puntuaciones de los jugadores.

Rejilla* Partida::rejilla_agua_ `[protected]`
Rejilla de la animación del agua.

Rejilla* Partida::rejilla_bunker_ `[protected]`
Rejilla de la animación del bunker.

Rejilla* Partida::rejilla_hojas_ `[protected]`
Rejilla de la animación de las hojas.

Teclado Partida::teclado_ `[protected]`
Teclado de la partida.

6.2.14. Clase Practicar

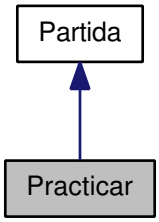


Figura 6.19: Diagrama de herencia de la clase Practicar

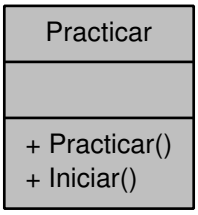


Figura 6.20: Detalle de la clase Practicar en UML

6.2.14.1. Descripción

Modo de juego de práctica.

Implementación del modo de juego practicar.

En una partida de práctica, uno o varios jugadores pueden practicar un determinado hoyo de un campo.

No existen ganadores en este modo de juego.

6.2.14.2. Documentación del constructor y destructor

Practicar::Practicar (std::vector< Datos::Jugador > & *jugadores*, Datos::Campo & *campo*, Menu * *menu*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con el menú y la pantalla.

Crea y asigna las bolas de golf y los jugadores.

Inicializa las puntuaciones a 0 y carga el campo.

Coloca las Bolas sobre el campo, actualiza la lista de bolas inactivas y activa el primer jugador.

Crea las animaciones de las colisiones de la bola con los elementos del campo.

Parámetros:

jugadores Datos de los jugadores que participan en la partida.

campo Datos del campo en el que se desarrollará la partida.

menu Menú del juego.

pantalla Pantalla del juego.

6.2.14.3. Documentación de las funciones miembro

int Practicar::Iniciar () `[virtual]`

Inicia la partida.

Devuelve:

- **0** hemos salido de la partida desde el menú de pausa.
- **-1** hemos cerrado la ventana del juego.

Implementa **Partida**.

6.2.15. Clase Rejilla

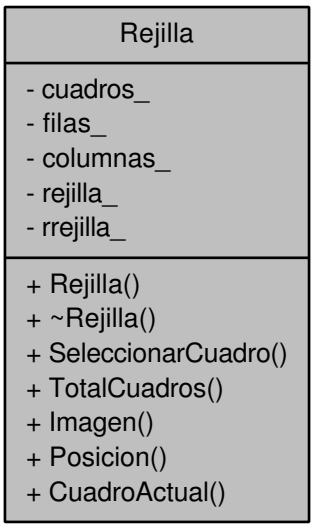


Figura 6.21: Detalle de la clase Rejilla en UML

6.2.15.1. Descripción

Rejilla de imágenes.

La clase rejilla permite el acceso a los diferentes cuadros que componen una rejilla de imágenes.

6.2.15.2. Documentación del constructor y destructor

Rejilla::Rejilla (std::string *ruta*, int *filas*, int *columnas*)

Constructor.

Carga la rejilla de imágenes, establece el número de filas y de columnas y selecciona un cuadro nulo.

Parámetros:

ruta Ruta de la rejilla que se quiere cargar.

filas Número de filas que posee la rejilla.

columnas Número de columnas que posee la rejilla.

Rejilla::~Rejilla ()

Destructor.

Destruye la imagen de la rejilla.

6.2.15.3. Documentación de las funciones miembro

int Rejilla::CuadroActual () const
Devuelve el cuadro de la rejilla que está seleccionado actualmente.

Devuelve: Cuadro de la rejilla.

SDL_Surface * Rejilla::Imagen ()
Devuelve la imagen del cuadro de la rejilla seleccionado.

Devuelve: Imagen del cuadro seleccionado actualmente.

SDL_Rect & Rejilla::Posicion ()
Devuelve la posición del cuadro de la rejilla seleccionado.

Devuelve: Posicion del cuadro seleccionado actualmente.

void Rejilla::SeleccionarCuadro (int *cuadro*)
Selecciona un cuadro de la rejilla.
Selecciona un determinado cuadro en la rejilla de imágenes.

Parámetros: ***cuadro*** Número de cuadro que se desea seleccionar.

int Rejilla::TotalCuadros () const
Devuelve el total de cuadros que posee la rejilla.

Devuelve: Número de cuadros de la rejilla.

6.2.15.4. Documentación de los datos miembro

int Rejilla::columnas_ `[private]`
Número de columnas de la rejilla.

int Rejilla::cuadro_ `[private]`
Cuadro de la rejilla seleccionado.

int Rejilla::filas_ `[private]`
Número de filas de la rejilla.

SDL_Surface* Rejilla::rejilla_ `[private]`

Imagen de la rejilla.

SDL_Rect Rejilla::rrejilla_ `[private]`

Posición de la rejilla.

6.2.16. Clase Strokeplay

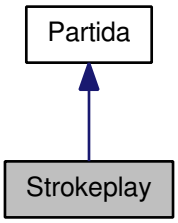


Figura 6.22: Diagrama de herencia de la clase Strokeplay

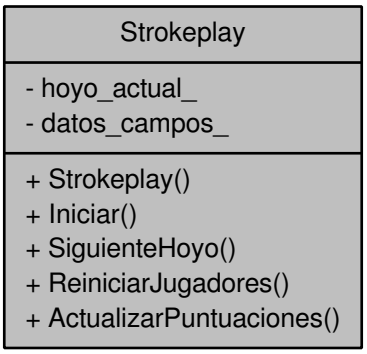


Figura 6.23: Detalle de la clase Strokeplay en UML

6.2.16.1. Descripción

Modo de juego Strokeplay.

Implementación del modo de juego Strokeplay.

En el modo de juego Strokeplay, uno o más jugadores se enfrentan en todos los hoyos de un campo.

Al final de cada hoyo, cada jugador obtendrá una puntuación determinada por la resta del par del hoyo menos el par del propio jugador.

Ganará la partida el jugador que obtenga la puntuación más baja, es decir, el jugador que haya completado todos los hoyos de un campo haciendo menor par que sus contrincantes.

6.2.16.2. Documentación del constructor y destructor

Strokeplay::Strokeplay (std::vector< Datos::Jugador > & *jugadores*, std::vector< Datos::Campo > & *campos*, Menu * *menu*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con el menú y la pantalla.

Crea y asigna las bolas de golf y los jugadores.

Inicializa las puntuaciones a 0 y carga el primer campo.

Coloca las Bolas sobre el campo, actualiza la lista de bolas inactivas y activa el primer jugador.

Crea las animaciones de las colisiones de la bola con los elementos del campo.

Parámetros:

jugadores Datos de los jugadores que participan en la partida.

campos Datos de los campos en los que se desarrollará la partida.

menu Menú del juego.

pantalla Pantalla del Juego.

6.2.16.3. Documentación de las funciones miembro

void Strokeplay::ActualizarPuntuaciones ()

Actualiza las puntuaciones.

Actualiza las puntuaciones de los jugadores.

La puntuación del hoyo se obtiene restando al par del jugador el par del campo.

int Strokeplay::Iniciar () `[virtual]`

Inicia la partida.

Devuelve:

- **0** hemos salido de la partida desde el menú de pausa.
- **-1** hemos cerrado la ventana del juego.

Implementa **Partida**.

void Strokeplay::ReiniciarJugadores ()

Reinicia todos los jugadores de la partida.

bool Strokeplay::SiguienteHoyo ()

Pasa al siguiente hoyo.

Pasa al siguiente hoyo, destruyendo el actual y modificando el campo activo.

Devuelve:

- **Verdadero** en caso de haber más hoyos.
- **Falso** en caso de no haber más hoyos.

6.2.16.4. Documentación de los datos miembro

std::vector<Datos::Campo> Strokeplay::datos_campos_ `[private]`

Datos de los campos de la partida.

unsigned int Strokeplay::hoyo_actual_ `[private]`

Hoyo en el que se está jugando.

6.2.17. Clase Teclado

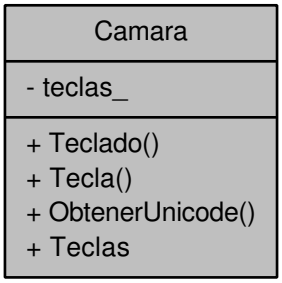


Figura 6.24: Detalle de la clase Teclado en UML

6.2.17.1. Descripción

Teclado del juego.

En esta clase se definen las teclas del juego y los métodos de control del teclado.

6.2.17.2. Documentación de las enumeraciones miembro de la clase

enum Teclado::Teclas

Teclas configuradas.

Valores de la enumeración:

PANTALLA_COMPLETA

ABAJO

ARRIBA

IZQUIERDA

DERECHA

ESPACIO

ESCAPE

ACEPTAR

INTRO

MAPA

PAUSA

BORRAR

N_TECLAS

6.2.17.3. Documentación del constructor y destructor

Teclado::Teclado ()

Constructor.

Realiza la asignación de teclas.

6.2.17.4. Documentación de las funciones miembro

char Teclado::ObtenerUnicode (const SDL_KeyboardEvent * *evento*)

Carácter unicode de una tecla.

Parámetros: ***evento*** Evento del teclado a procesar.

Devuelve: Caracter unicode de la tecla pulsada.

SDLKey & Teclado::Tecla (int *tecla*)

Devuelve la tecla asignada.

Parámetros: ***tecla*** Tecla que queremos obtener.

Devuelve: Tecla asignada.

6.2.17.5. Documentación de los datos miembro

SDLKey Teclado::teclas_[N_TECLAS] `[private]`

Teclas configuradas.

6.3. Capa de presentación

6.3.1. Clase Interfaz

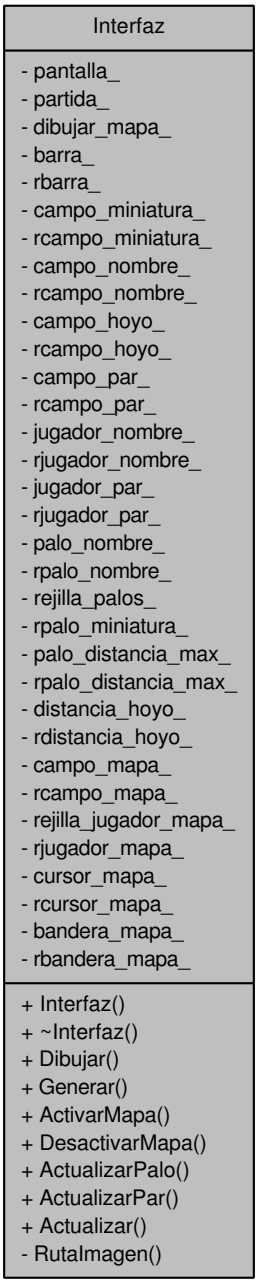


Figura 6.25: Detalle de la clase Interfaz en UML

6.3.1.1. Descripción

Interfaz del juego.

La interfaz informa al jugador sobre los datos más importantes de la partida.

En ella, se indica el nombre del campo y el número de hoyo en el que se está jugando, así como el par de éste último.

Por otra parte, se muestra el nombre y el par del jugador activo, el nombre y el alcance del palo de golf que tiene seleccionado y la distancia a la que se encuentra el jugador con respecto al hoyo.

También permite al jugador visualizar el mapa del campo en el que se está jugando.

6.3.1.2. Documentación del constructor y destructor

Interfaz::Interfaz (Partida * *partida*, Pantalla * *pantalla*)

Constructor.

Realiza las asociaciones con la partida y la pantalla.

Establece los punteros de las imágenes como nulos y desactiva el mapa del campo. Crea la rejilla de imágenes para la representación de los jugadores en el mapa en función de su color.

Parámetros:

partida Partida que se está desarrollando.

pantalla Pantalla donde se dibujará la interfaz.

Interfaz::~Interfaz ()

Destructor.

Destruye todas las imágenes de la interfaz y la rejilla de los jugadores en el mapa.

6.3.1.3. Documentación de las funciones miembro

void Interfaz::ActivarMapa ()

Activa el mapa del campo.

Activa el mapa para que sea dibujado.

void Interfaz::Actualizar ()

Actualiza todos los datos e imágenes de la interfaz, salvo el nombre del campo, el número de hoyo y la imagen en miniatura del campo.

void Interfaz::ActualizarPalo ()

Actualiza la imagen, el nombre y el alcance del palo seleccionado.

void Interfaz::ActualizarPar ()

Actualiza el par del jugador.

void Interfaz::DesactivarMapa ()

Desactiva el mapa del campo.

Desactiva el mapa para que no sea dibujado.

void Interfaz::Dibujar ()

Dibuja la interfaz.

Dibuja la interfaz en la pantalla y actualiza la posición del mapa en función de la posición del jugador.

void Interfaz::Generar ()

Genera la interfaz.

Genera la interfaz por completo, carga todas las imágenes y genera todos los datos.

std::string Interfaz::RutaImagen (const char * *imagen*) `[private]`

Genera la ruta a una imagen.

Genera una ruta desde el directorio de imágenes de la interfaz.

Parámetros: ***imagen*** Nombre de la imagen.

Devuelve: Ruta a la imagen.

6.3.1.4. Documentación de los datos miembro

SDL_Surface* Interfaz::bandera_mapa_ `[private]`

Imagen de la bandera en el mapa.

SDL_Surface* Interfaz::barra_ `[private]`

Imagen de la barra de la interfaz.

SDL_Surface* Interfaz::campo_hoyo_ `[private]`

Imagen del número de hoyo.

SDL_Surface* Interfaz::campo_mapa_ `[private]`

Imagen del mapa del campo.

SDL_Surface* Interfaz::campo_miniatura_ `[private]`
Imagen de la miniatura del campo.

SDL_Surface* Interfaz::campo_nombre_ `[private]`
Imagen del nombre del campo.

SDL_Surface* Interfaz::campo_par_ `[private]`
Imagen del par del hoyo.

SDL_Surface* Interfaz::cursor_mapa_ `[private]`
Imagen del cursor en el mapa.

bool Interfaz::dibujar_mapa_ `[private]`
Variable de control para dibujar el mapa.

SDL_Surface* Interfaz::distancia_hoyo_ `[private]`
Imagen de la distancia al hoyo.

SDL_Surface* Interfaz::jugador_nombre_ `[private]`
Imagen del nombre del jugador.

SDL_Surface* Interfaz::jugador_par_ `[private]`
Imagen del par del jugador.

SDL_Surface* Interfaz::palo_distancia_max_ `[private]`
Imagen del alcance del palo.

SDL_Surface* Interfaz::palo_nombre_ `[private]`
Imagen del nombre del palo.

Pantalla* Interfaz::pantalla_ `[private]`
Pantalla del juego.

Partida* Interfaz::partida_ `[private]`
Partida que está actualmente en juego.

SDL_Rect Interfaz::rbandera_mapa_ `[private]`
Posición de la bandera en el mapa.

SDL_Rect Interfaz::rbarra_ `[private]`
Posición de la barra de la interfaz.

SDL_Rect Interfaz::rcampo_hoyo_ `[private]`
Posición del número de hoyo.

SDL_Rect Interfaz::rcampo_mapa_ `[private]`
Posición del mapa del campo.

SDL_Rect Interfaz::rcampo_miniatura_ `[private]`
Posición de la miniatura del campo.

SDL_Rect Interfaz::rcampo_nombre_ `[private]`
Posición del nombre del campo.

SDL_Rect Interfaz::rcampo_par_ `[private]`
Posición del par del hoyo.

SDL_Rect Interfaz::rcursor_mapa_ `[private]`
Posición del cursor en el mapa.

SDL_Rect Interfaz::rdistancia_hoyo_ `[private]`
Posición de la distancia al hoyo.

Rejilla∗ Interfaz::rejilla_jugador_mapa_ `[private]`
Rejilla del jugador en el mapa.

Rejilla∗ Interfaz::rejilla_palos_ `[private]`
Rejilla de imágenes de los palos.

SDL_Rect Interfaz::rjugador_mapa_ `[private]`
Posición del jugador en el mapa.

SDL_Rect Interfaz::rjugador_nombre_ `[private]`

Posición del nombre del jugador.

SDL_Rect Interfaz::rjugador_par_ `[private]`

Posición del par del jugador.

SDL_Rect Interfaz::rpalo_distancia_max_ `[private]`

Posición del alcance del palo.

SDL_Rect Interfaz::rpalo_miniatura_ `[private]`

Posición de la imagen del palo.

SDL_Rect Interfaz::rpalo_nombre_ `[private]`

Posición del nombre del palo.

6.3.2. Clase Menú

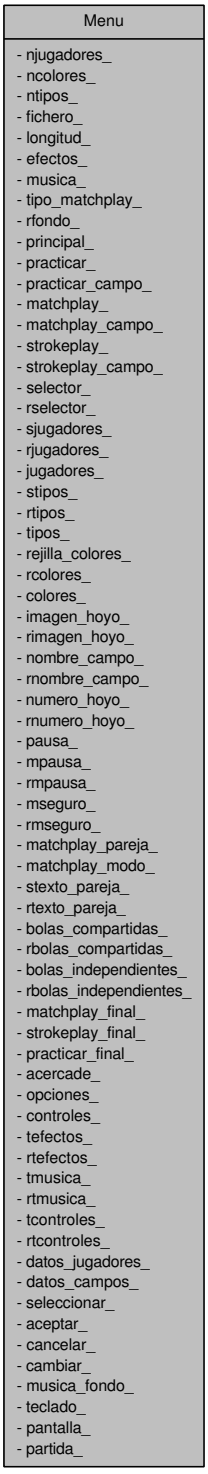


Figura 6.26: Detalle de la clase Menú en UML (Primera parte)

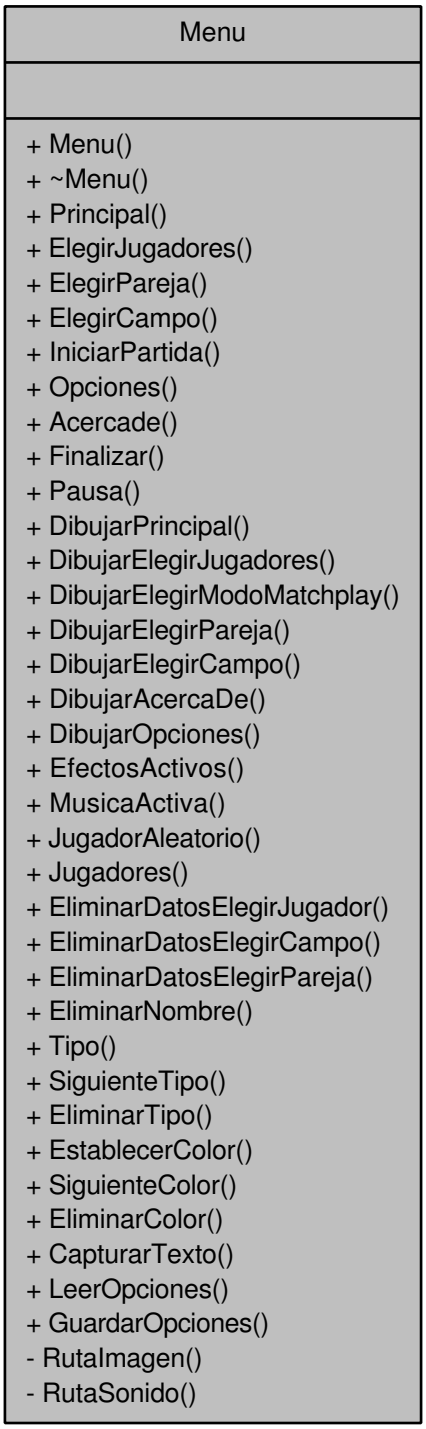


Figura 6.27: Detalle de la clase Menú en UML (Segunda parte)

6.3.2.1. Descripción

Menú del juego.

El menú del juego permite al jugador seleccionar los diferentes modos de juego, las opciones, obtener información sobre SDLgolf o salir de la aplicación. Esta clase incluye

todos los submenús, como los de elección de jugador, elección de campo, pantallas de resumen y menú de pausa.

6.3.2.2. Documentación de las enumeraciones miembro de la clase

enum Menu::Modos

Modos de juego disponibles.

Valores de la enumeración:

PRACTICAR

MATCHPLAY

STROKEPLAY

6.3.2.3. Documentación del constructor y destructor

Menu::Menu (Pantalla * *pantalla*)

Constructor.

Carga las imágenes principales del menú del juego, la música, los efectos de sonido y el fichero de nombres.

Parámetros: ***pantalla*** Pantalla del juego.

Menu::~Menu ()

Destructor.

Destruye todas las imágenes, rejillas y el vector del fichero de nombres.

6.3.2.4. Documentación de las funciones miembro

void Menu::Acercade ()

Muestra el menú de información.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **0** En otro caso.

int Menu::CapturarTexto (int *jugador*, int *modo*)

Captura las pulsaciones del teclado.

Captura texto del teclado y genera la imagen del nombre del jugador.

Parámetros:

jugador Número de jugador.

modo Modo de juego.

Devuelve:

- **1** Si se ha cerrado la ventana del juego.
- **0** En otro caso.

void Menu::DibujarAcercaDe () const

Dibuja el menú de acerca de.

void Menu::DibujarElegirCampo (int *modo*) const

Dibuja el menú de elección de campo.

Parámetros: ***modo*** Modo de juego.

void Menu::DibujarElegirJugadores (int *modo*) const

Dibuja el menú de elección de jugadores.

Parámetros: ***modo*** Modo de juego.

void Menu::DibujarElegirModoMatchplay () const

Dibuja el menú de elección de modo de Matchplay.

void Menu::DibujarElegirPareja () const

Dibuja el menú de elección de pareja.

void Menu::DibujarOpciones () const

Dibuja el menú de opciones.

void Menu::DibujarPrincipal () const

Dibuja el menú principal.

bool Menu::EfectosActivos () const

Indica el estado de los efectos de sonido.

Devuelve:

- **Verdadero** si los efectos de sonido están activados.
- **Falso** si los efectos de sonido están desactivados.

int Menu::ElegirCampo (int *modo*)

Muestra el menú de elección de campo.

Parámetros: ***modo*** Modo de juego.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **1** Si se ha pulsado el botón de volver.
- **0** En otro caso.

int Menu::ElegirJugadores (int *modo*)

Muestra el menú de elección de jugadores.

Parámetros: ***modo*** Modo de juego.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **0** En otro caso.

int Menu::ElegirPareja ()

Muestra el menú de elección de pareja para el modo Matchplay.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **1** Si se ha pulsado el botón de volver.
- **0** En otro caso.

void Menu::EliminarColor (int *jugador*)

Elimina la imagen del color del jugador.

Parámetros: ***jugador*** Número de jugador.

void Menu::EliminarDatosElegirCampo ()
Elimina datos de la elección de campo.
Elimina la imagen del campo, el nombre del campo y el número de hoyo.

void Menu::EliminarDatosElegirJugador ()
Elimina datos de la elección de jugador.
Elimina las imágenes de los nombres, tipos y colores de los jugadores creados.

void Menu::EliminarDatosElegirPareja ()
Elimina las imágenes de los textos.

void Menu::EliminarNombre (int *jugador*)
Elimina la imagen del nombre del jugador.

Parámetros: ***jugador*** Número de jugador.

void Menu::EliminarTipo (int *jugador*)
Elimina la imagen del tipo del jugador.

Parámetros: ***jugador*** Número de jugador.

void Menu::EstablecerColor (int *jugador*, int *color*)
Selecciona de la rejilla de imágenes el color del jugador.

Parámetros:

jugador Número de jugador.
color Color del jugador.

int Menu::Finalizar (int *modo*)
Muestra el menú de finalización de partida.

Parámetros: ***modo*** Modo de juego.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **0** En otro caso.

void Menu::GuardarOpciones ()

Guarda las opciones.

Guarda las opciones del juego en el fichero de configuración.

int Menu::IniciarPartida (int *modo*)

Inicia la partida en función del modo de juego.

Parámetros: ***modo*** Modo de juego.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **1** Si se ha pulsado el botón de volver.
- **0** En otro caso.

void Menu::JugadorAleatorio (int *jugador*, int *modo*)

Genera un nombre aleatorio para el jugador indicado.

El nombre es seleccionado de una lista de nombres cargada previamente en memoria.

Parámetros:

jugador Número de jugador.

modo Modo de juego.

int Menu::Jugadores () const

Número de jugadores.

Devuelve: Número de jugadores creados que participarán en la partida.

void Menu::LeerOpciones ()

Lee las opciones.

Lee las opciones del juego del fichero de configuración y establece los parámetros.

bool Menu::MusicaActiva () const

Indica el estado de la música.

Devuelve:

- **Verdadero** si la música está activada.
- **Falso** si la música está desactivada.

int Menu::Opciones ()

Muestra el menú de opciones.

Devuelve:

- **-1** Si se ha cerrado la ventana del juego.
- **0** En otro caso.

int Menu::Pausa ()

Ejecuta el menú de pausa.

Devuelve:

- **-1** Si se sale de la partida.
- **1** Si se ha cerrado la ventana del juego.
- **0** Si se continua la partida.

int Menu::Principal ()

Muestra el menú principal del juego.

Devuelve: **0** Si se sale del juego o se cierra la ventana.

std::string Menu::RutaImagen (const char * *imagen*) `[private]`

Genera ruta a imagen.

Genera una ruta desde el directorio de imágenes del menú.

Parámetros: ***imagen*** Nombre de la imagen.

Devuelve: Ruta a la imagen.

std::string Menu::RutaSonido (const char * *sonido*) `[private]`

Genera ruta a sonido.

Genera una ruta a un sonido desde el directorio de efectos de sonido.

Parámetros: ***imagen*** Nombre del sonido.

Devuelve: Ruta al sonido.

void Menu::SiguienteColor (int *jugador*)

Establece el siguiente color para el jugador.

Parámetros: ***jugador*** Número de jugador.

void Menu::SiguienteTipo (int *jugador*)

Establece el siguiente tipo para el jugador.

Parámetros: ***jugador*** Número de jugador.

void Menu::Tipo (int *jugador*, int *tipo*)

Crea la imagen del tipo del jugador.

Parámetros:

jugador Número de jugador.

tipo Tipo de jugador.

6.3.2.5. Documentación de los datos miembro

Sonido Menu::aceptar_ `[private]`

una opción.

Sonido que se produce al aceptar

SDL_Surface* Menu::acercade_ `[private]`

Imagen del menú acercade.

SDL_Surface* Menu::bolas_compartidas_ `[private]`

Imagen del texto de bolas compartidas.

SDL_Surface* Menu::bolas_independientes_ `[private]`

Imagen del texto de bolas independientes.

Sonido Menu::cambiar_ `[private]`

Sonido que se produce al cambiar una opción.

Sonido Menu::cancelar_ [private]
una opción.
Sonido que se produce al cancelar

std::vector<int> Menu::colores_ [private]
Cadenas de texto con los colores de los jugadores.

SDL_Surface* Menu::controles_ [private]
Imagen de los controles del juego.

std::vector<Datos::Campo> Menu::datos_campos_ [private]
la partida.
Datos de los campos en los que se jugará

std::vector<Datos::Jugador> Menu::datos_jugadores_ [private]
Datos de los jugadores que participarán en la partida.

bool Menu::efectos_ [private]
Estado de los efectos de sonido.

char* Menu::fichero_ [private]
Fichero de nombres para la generación aleatoria.

SDL_Surface* Menu::imagen_hoyo_ [private]
Imagen del campo.

std::vector<std::string> Menu::jugadores_ [private]
Cadenas de texto con los nombres de los jugadores.

int Menu::longitud_ [private]
Longitud del fichero.

SDL_Surface* Menu::matchplay_ [private]
Imagen del menú matchplay.

SDL_Surface* Menu::matchplay_campo_ `[private]`
Imagen del menú de elección de campo del modo matchplay.

SDL_Surface* Menu::matchplay_final_ `[private]`
Imagen del menú de finalización de Matchplay.

SDL_Surface* Menu::matchplay_modo_ `[private]`
Imagen del menú de tipo del modo matchplay.

SDL_Surface* Menu::matchplay_pareja_ `[private]`
Imagen del menú de elección de pareja del modo matchplay.

SDL_Surface* Menu::mpausa_ `[private]`
Imagen del cuadro de opciones menú de pausa.

SDL_Surface* Menu::mseguro_ `[private]`
menú de pausa.
Imagen del cuadro de confirmación del

bool Menu::musica_ `[private]`
Estado de la música del juego.

Musica Menu::musica_fondo_ `[private]`
Música de fondo.

int Menu::ncolores_ `[private]`
Número de colores.

int Menu::njugadores_ `[private]`
Número de jugadores.

SDL_Surface* Menu::nombre_campo_ `[private]`
Imagen del nombre del campo.

int Menu::ntipos_ [private]
Número de tipos.

SDL_Surface* Menu::numero_hoyo_ [private]
Posición del numero de hoyo del campo.

SDL_Surface* Menu::opciones_ [private]
Imagen del menú de opciones.

Pantalla* Menu::pantalla_ [private]
Pantalla del juego.

Partida* Menu::partida_ [private]
Partida que se creará.

SDL_Surface* Menu::pausa_ [private]
Imagen del menú de pausa.

SDL_Surface* Menu::practicar_ [private]
Imagen del menú practicar.

SDL_Surface* Menu::practicar_campo_ [private]
Imagen del menú de elección de campo del modo practicar.

SDL_Surface* Menu::practicar_final_ [private]
Imagen del menú de finalización de Practicar.

SDL_Surface* Menu::principal_ [private]
Imagen del menú principal.

SDL_Rect Menu::rbolas_compartidas_ [private]
Posición de la imagen del texto de bolas compartidas.

SDL_Rect Menu::rbolas_independientes_ [private]
Posición de la imagen del texto de bolas independientes.

std::vector<SDL_Rect> Menu::rcolores_ `[private]`
Posición de las imágenes de los colores de los jugadores.

Rejilla* Menu::rejilla_colores_ `[private]`
Imágenes de los colores de los jugadores.

SDL_Rect Menu::rfondo_ `[private]`
Posición de las imágenes de fondo del menú.

SDL_Rect Menu::rimagen_hoyo_ `[private]`
Posición de la imagen del campo.

std::vector<SDL_Rect> Menu::rjugadores_ `[private]`
Posición de las imágenes de los nombres de los jugadores.

SDL_Rect Menu::rmpausa_ `[private]`
Posición del cuadro de opciones del menú de pausa.

SDL_Rect Menu::rmseguro_ `[private]`
Posición del cuadro de confirmación del menú de pausa.

SDL_Rect Menu::rnombre_campo_ `[private]`
Posición del nombre del campo.

SDL_Rect Menu::rnumero_hoyo_ `[private]`
Posición del nombre del campo.

SDL_Rect Menu::rselector_ `[private]`
Posición del selector del menú.

SDL_Rect Menu::rtcontroles_ `[private]`
Posición del texto de los controles del juego.

SDL_Rect Menu::rtefectos_ `[private]`
Posición de la imagen del texto del estado de los efectos de sonido.

std::vector<SDL_Rect> Menu::rtexto_pareja_ [private]
Posición de las imágenes de los nombres de las parejas.

std::vector<SDL_Rect> Menu::rtipos_ [private]
Posición de las imágenes de los tipos de los jugadores.

SDL_Rect Menu::rtmusica_ [private]
Posición de la imagen del texto del estado de la música del juego.

Sonido Menu::seleccionar_ [private]
Sonido que se produce al seleccionar una opción.

SDL_Surface* Menu::selector_ [private]
Imagen del selector del menú.

std::vector<SDL_Surface *> Menu::sjugadores_ [private]
Imágenes de los nombres de los jugadores.

std::vector<SDL_Surface *> Menu::stexto_pareja_ [private]
Imágenes de los nombres de las parejas.

std::vector<SDL_Surface *> Menu::stipos_ [private]
Imágenes de los tipos de los jugadores.

SDL_Surface* Menu::strokeplay_ [private]
Imagen del menú strokeplay.

SDL_Surface* Menu::strokeplay_campo_ [private]
Imagen del menú de elección de campo del modo strokeplay.

SDL_Surface* Menu::strokeplay_final_ [private]
Imagen del menú de finalización de Strokeplay.

SDL_Surface* Menu::tcontroles_ [private]
Imagen del texto de los controles del juego.

Teclado Menu::teclado_ `[private]`

Teclado del menú.

SDL_Surface∗ Menu::tefectos_ `[private]`

Imagen del texto del estado de los efectos de sonido.

int Menu::tipo_matchplay_ `[private]`

Tipo de Matchplay que se va a jugar.

std::vector<int> Menu::tipos_ `[private]`

Cadenas de texto con los nombres de los jugadores.

SDL_Surface∗ Menu::tmusica_ `[private]`

Imagen del texto del estado de la música del juego.

6.3.3. Clase Música

Figura 6.28: Detalle de la clase Música en UML

6.3.3.1. Descripción

Música del juego.

La clase música permite dotar de música de fondo al menú del juego y generar sonido ambiental en la partida que esté en curso.

6.3.3.2. Documentación del constructor y destructor

Musica::Musica ()

Constructor.

Establece el puntero a la canción como nulo.

Musica::∼Musica ()

Destructor.

Destruye la música que se ha cargado.

6.3.3.3. Documentación de las funciones miembro

void Musica::Cargar (std::string *ruta*)

Carga el fichero de música.

Parámetros: ***ruta*** Ruta al fichero de música.

bool Musica::Reproduciendo () const

Función observadora.

Devuelve:

- **Verdadero** Si la música se está reproduciendo.
- **Falso** Si la música no se está reproduciendo.

void Musica::Reproducir (int *veces*) const

Reproduce el fichero de música.

Parámetros:

veces
- **-1** Reproduce la música constantemente.
- **n** Reproduce la música *n* veces.

6.3.3.4. Documentación de los datos miembro

Mix_Music∗ Musica::musica_ `[private]`

Música cargada.

6.3.4. Clase Pantalla

Figura 6.29: Detalle de la clase Pantalla en UML

6.3.4.1. Descripción detallada

Pantalla del juego.

En la pantalla del juego se representarán todos los elementos gráficos del mismo. Es uno de los elementos vitales del juego.

6.3.4.2. Documentación del constructor y destructor

Pantalla::Pantalla ()

Constructor.

Crea la superficie de la pantalla del juego, oculta el cursor, muestra el nombre del juego en el borde de la ventana y crea la cámara.

Pantalla::∼Pantalla ()

Destructor.

Destruye la pantalla y la cámara.

6.3.4.3. Documentación de las funciones miembro

void Pantalla::Actualizar ()

Actualiza la pantalla.

int Pantalla::Alto () const

Alto de la pantalla.

Devuelve: Alto de la pantalla.

int Pantalla::Ancho () const

Ancho de la pantalla.

Devuelve: Ancho de la pantalla.

void Pantalla::CentrarCamara (const SDL_Rect & *objetivo*, const SDL_Rect & *superficie*)

Centra la cámara en un determinado objetivo.

void Pantalla::Copiar (SDL_Surface ∗∗ *destino*)

Realiza una copia de la superficie de la pantalla.

Parámetros: ***destino*** Superficie donde se va a copiar la pantalla.

void Pantalla::Dibujar (const SDL_Surface ∗ *origen*, const SDL_Rect ∗ *rorigen*, const SDL_Rect ∗ *rdestino*)

Dibuja en pantalla.

Dibuja sobre la superficie de la pantalla tomando las coordenadas relativas de la cámara.

Parámetros:

origen Imagen que se va a dibujar.

rorigen Rectángulo de la imagen que se va a dibujar.

rdestino Coordenadas donde se va a dibujar la imagen.

void Pantalla::Dibujar (const SDL_Surface * *origen*, const SDL_Rect * *rdestino*)

Dibuja en pantalla.

Dibuja sobre la superficie de la pantalla tomando las coordenadas relativas de la cámara.

Parámetros:

origen Imagen que se va a dibujar.

rdestino Coordenadas donde se va a dibujar la imagen.

void Pantalla::Dibujar (const SDL_Surface * *origen*)

Dibuja en pantalla.

Dibuja sobre la superficie de la pantalla tomando las coordenadas relativas de la cámara.

Parámetros: ***origen*** Imagen que se va a dibujar.

void Pantalla::DibujarDirectamente (const SDL_Surface * *origen*, const SDL_Rect * *rorigen*, const SDL_Rect * *rdestino*)

Dibuja directamente en pantalla.

Dibuja sobre la superficie de la pantalla tomando las coordenadas directamente.

Parámetros:

origen Imagen que se va a dibujar.

rorigen Rectángulo de la imagen que se va a dibujar.

rdestino Coordenadas donde se va a dibujar la imagen.

void Pantalla::DibujarDirectamente (const SDL_Surface * *origen*, const SDL_Rect * *destino*)

Dibuja directamente en pantalla.

Dibuja sobre la superficie de la pantalla tomando las coordenadas directamente.

Parámetros:

origen Imagen que se va a dibujar.

destino Coordenadas donde se va a dibujar la imagen.

void Pantalla::PantallaCompleta ()

Establece el modo de pantalla completa.

6.3.4.4. Documentación de los datos miembro

Camara* Pantalla::camara_ `[private]`

Cámara de la pantalla.

SDL_Surface* Pantalla::pantalla_ `[private]`

Superficie de la pantalla.

SDL_Rect Pantalla::rpantalla_ `[private]`

Posición de la pantalla.

6.3.5. Clase Sonido

Figura 6.30: Detalle de la clase Sonido en UML

6.3.5.1. Descripción

Efectos de sonido.

Esta clase se encarga de generar los efectos de sonido que se producen en el juego, tales como la colisión de la bola con los elementos del campo, los efectos de sonido del menú, el efecto de sonido que produce el palo al impactar contra la bola, etc.

6.3.5.2. Documentación del constructor y destructor

Sonido::Sonido ()

Constructor.

Establece el puntero al sonido como nulo.

Sonido::~Sonido ()

Destructor.

Destruye el efecto de sonido.

6.3.5.3. Documentación de las funciones miembro

void Sonido::Cargar (std::string *ruta*)
Carga el fichero de sonido.

Parámetros: ***ruta*** Ruta al fichero de sonido.

void Sonido::Reproducir ()
Reproduce el efecto de sonido.

6.3.5.4. Documentación de los datos miembro

Mix_Chunk* Sonido::sonido_ `[private]`
Efecto de sonido.

Capítulo 7

Implementación

Desde las primeras fases del desarrollo de SDLgolf y hasta su finalización, fueron apareciendo, paulatinamente, diversos problemas y dificultades que tuve que ir solventando de la mejor forma que sabía.

Según avanzaba en el proyecto, el número de problemas que surgían era cada vez mayor, así como el número de cosas que debía hacer.

El conjunto de bibliotecas SDL proporciona unas operaciones muy básicas y a la vez muy potentes para dibujar en 2D. Sin embargo, desde un punto de vista práctico, el trabajar a tan bajo nivel se traduce a la necesidad de crear clases y funciones específicas para cada uno de los aspectos y necesidades del videojuego que se está desarrollando, lo que a su vez significa que es necesario implementar una estructura funcional y lógica a partir de la cual sea posible controlar desde una simple carga de imágenes, hasta la detección de colisiones, pasando por el motor físico del juego o la creación de animaciones.

7.1. Bola de golf

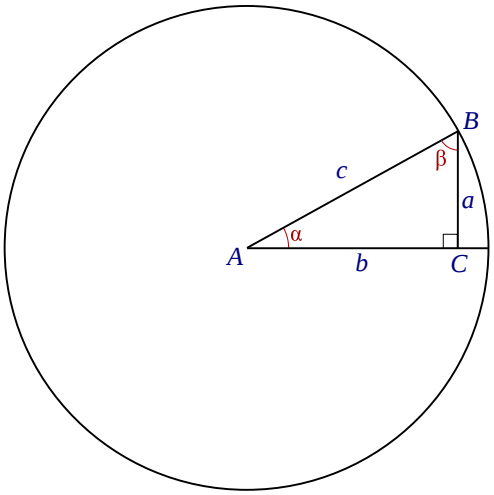
La bola de golf es el elemento clave del juego y en ella fue donde inicialmente centré mi atención. Era de vital importancia conseguir que el jugador tuviese la sensación de que la bola adquiría altura y se desplazaba realmente por la superficie de juego.

En este punto jugaba un papel muy importante la proyección de su sombra en el terreno. Cuando la bola está en movimiento, su sombra se desplaza en línea recta desde la posición de salida hasta la posición de destino, mientras que la imagen de la bola describe una parábola a lo largo de todo el recorrido. De esta forma, se consigue transmitir al jugador la sensación de profundidad y altura que tan importantes son en este videojuego.

El desplazamiento de la bola queda determinado por un triángulo rectángulo cuya hipotenusa es la distancia euclídea (ver 7.1) entre la posición de la bola y el punto de destino. Este último puede calcularse aplicando las razones trigonométricas del triángulo rectángulo (ver 7.2) y conociendo el alcance máximo del palo de golf.

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figura 7.1: Distancia euclidiana entre los puntos P1(*x1, y1*) y P2(*x2, y2*)



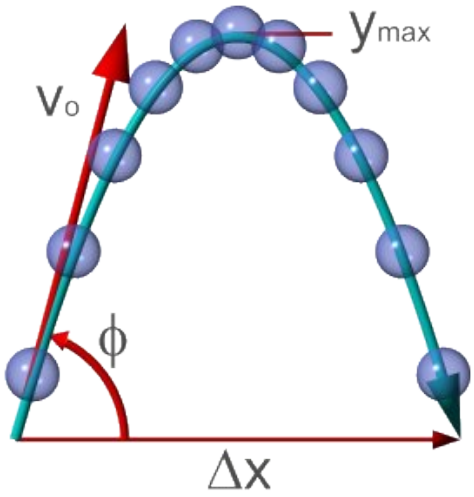
$$\sin(\alpha) = \frac{a}{c}$$

$$\cos(\alpha) = \frac{b}{c}$$

$$\tan(\alpha) = \frac{a}{b}$$

Figura 7.2: Razones trigonométricas del triángulo rectángulo.

Para el movimiento de la imagen de la bola y la sombra, se emplean las fórmulas del tiro parabólico (ver 7.3). Sabiendo el ángulo de salida que aplica el palo de golf y la distancia a recorrer, determinada por la hipotenusa del triángulo de desplazamiento, puede calcularse el tiempo. El cociente de la división de este último entre la longitud de la hipotenusa determina el incremento del tiempo para cada iteración del bucle que calcula los diferentes puntos del desplazamiento de la bola de golf. La coordenada de posición en el eje de abscisas para la imagen de la bola y la coordenada de la imagen de la sombra en el mismo eje, siempre serán iguales entre sí.



$$v = \begin{cases} v_{0x} = v_0 \cos\theta_0 \\ v_{0y} = v_0 \sin\theta_0 - gt \end{cases}$$

$$r = \begin{cases} x = v_0 t \cos\theta_0 \\ y = y_0 + v_0 t \sin\theta_0 - \frac{1}{2} g t^2 \end{cases}$$

Figura 7.3: Ecuaciones del tiro parabólico.

Antes de llevarse a cabo del lanzamiento, el jugador puede determinar el nivel de fuerza y el efecto que quiere darle al desplazamiento de la bola.

El nivel de fuerza aplica un porcentaje sobre el alcance máximo del palo de golf, lo que repercute en la longitud de la hipotenusa del rectángulo de desplazamiento y ésta a su vez en la distancia final que recorrerá la bola.

El nivel de efecto aporta una curvatura en el desplazamiento de la bola. Dicha curvatura viene determinada por incrementos sinusoidales en el eje de ordenadas de la posición de la imagen de la sombra, y como consecuencia, también en el eje de ordenadas de la imagen de la bola. El cociente de la división del número π entre la longitud de la hipotenusa del triángulo de desplazamiento determina los incrementos de la curvatura para cada una de las iteraciones del bucle que calcula los diferentes puntos del desplazamiento de la bola de golf.

La altura a la que se encuentra la bola de golf en un determinado instante viene determinada por la diferencia entre la coordenada del eje de ordenadas de la sombra y la coordenada del eje de ordenadas de la bola.

Cuando la altura a la que se encuentra la bola se anula significa que dicha bola ha impactado contra la superficie de juego. En ese momento se decrementa su velocidad en un porcentaje. Si la velocidad no es nula, la bola rebotará, describiendo el mismo

comportamiento que en la salida pero con menor velocidad y por lo tanto, recorrerá menos distancia y alcanzará una altura menor. La distancia que recorra la bola en el rebote vendrá determinada por un porcentaje sobre la longitud de la hipotenusa del rectángulo de desplazamiento.

Al volver a impactar contra la superficie de juego, volverá a reducirse la velocidad, y si ésta sigue sin ser nula, la bola de golf rodará hasta detenerse, simulando el rozamiento. Como en el punto anterior, la distancia que recorra la bola mientras esté rodando vendrá determinada por un porcentaje sobre la longitud de la hipotenusa del rectángulo de desplazamiento.

7.2. Colisiones

La detección de colisiones es una de las necesidades básicas para el funcionamiento de cualquier videojuego.

En SDLgolf hemos de ser capaces de determinar cuándo la bola impacta contra los diferentes elementos que conforman el campo de golf.

La detección de una colisión lleva implícita el procesamiento de la misma. Por ejemplo, cuando la bola de golf cae al agua, se dibuja una animación en la que se aprecian las salpicaduras producidas por el impacto y se restaura la bola a la zona de tierra (de la trayectoria descrita en el desplazamiento) más próxima al lugar del impacto.

Inicialmente, empleé una técnica clásica para la detección de colisiones, que consiste en circunscribir en rectángulos todos los elementos que puedan participar en una colisión, esto es, la bola de golf y todos los elementos del campo, guardando las coordenadas de la posición de dichos rectángulos, así como el valor de la altura y la anchura para cada uno de ellos.

Con estos datos, es posible determinar cuándo dos rectángulos se superponen, lo que vendría a significar que se ha producido una colisión.

Pronto me percaté de que esa opción era inviable para SDLgolf. Para cada campo era necesario generar, de forma manual, una lista con todos los rectángulos de colisión y todos sus atributos. Era algo extremadamente tedioso y con un dinamismo nulo.

Pensando en una alternativa que se ajustara a las necesidades del videojuego que estaba desarrollando, se me ocurrió basar la detección de colisiones en los colores de los píxeles. Para ello, compruebo el color del píxel de la superficie del campo que se corresponde con el centro de la posición de la bola. En caso de que el color del píxel pertenezca a uno de los colores de colisión de los elementos del campo (agua, búnker o árboles), la colisión es detectada y procesada.

Cada campo puede tener sus propios colores de colisión. Los colores de colisión se almacenan en un fichero XML (ver Formato de los documentos XML) y se establecen cuando se crea el campo. De esta forma se consigue un dinamismo total, al no ser necesario preocuparse por las zonas de colisión, ya que es el propio juego quien se encarga de interpretar las zonas en función de los colores de los elementos.

7.3. Inteligencia Artificial

El desarrollo de una inteligencia artificial aceptable era necesario para el correcto funcionamiento de SDLgolf.

En un principio, pensé en que se podían predeterminar ciertas zonas de lanzamiento para que el ordenador, de forma aleatoria, eligiese cualquiera de ellas y fuese acercándose poco a poco al hoyo salvando los obstáculos. Sin embargo, esto suponía que el propio diseñador fuese el encargado de llevar a cabo la interpretación del terreno, algo que carecía de sentido si hablamos de inteligencia artificial.

Por tanto, era necesario que fuese el propio ordenador el encargado de interpretar los diferentes hoyos de juego de forma dinámica.

Para ello, el ordenador lleva a cabo una serie de simulaciones y va guardando todas las posibles soluciones en una lista.

En primer lugar, la computadora orienta el cursor del jugador al hoyo del campo y va simulando lanzamientos con todos los palos y a todos los niveles de fuerza posibles. Todos aquellos lanzamientos en los que no se produzca colisión se irán guardando en una lista.

A continuación, el ordenador operará de forma muy parecida. Comprobará todos los lanzamientos con todos los a palos y todos los niveles de fuerza para todos los ángulos posibles entre la posición del cursor orientado al hoyo hasta el límite del cuadrante. Es decir, el ordenador irá girando el cursor a la izquierda y para cada giro realizará la comprobación indicada anteriormente. De la misma forma que en el apartado anterior, todos aquellos lanzamientos en los que no se produzca colisión se irán guardando en otra lista.

Por último, el ordenador operará de forma exactamente igual, pero esta vez, girando el cursor a la derecha.

Al finalizar todas las simulaciones, tendremos tres listas de soluciones.

La inteligencia artificial de dificultad alta optará por la solución que más se aproxime al hoyo del campo.

Para la inteligencia artificial de nivel bajo y medio, se aplica un porcentaje (mayor en la dificultad baja) sobre la diferencia entre la distancia al hoyo de la peor solución, menos la distancia al hoyo de la mejor solución y se suma a la distancia al hoyo de la mejor solución.

Después, la computadora busca en la lista de soluciones aquella que su distancia se aproxime más a la determinada anteriormente y la elegirá como solución definitiva.

De esta forma, se consigue que las inteligencias artificiales de nivel medio y bajo se comporten de forma natural y realicen los hoyos con un mayor par.

Curiosamente y para mi sorpresa, fue más difícil diseñar la inteligencia artificial de dificultad media y baja que la dificultad alta.

Capítulo 8

Pruebas

La realización de pruebas de software, es una de las fases del ciclo del software y permiten la identificación de posibles fallos en la implementación, calidad o usabilidad de la aplicación.

8.1. Plan de pruebas

Debido a la arquitectura de SDLgolf y a la metodología basada en componentes que se ha seguido en el desarrollo, las diferentes partes han ido siendo probadas de forma independiente. Aun así, el videojuego ha sido sometido a las siguientes pruebas:

Pruebas de configuración:

- El modo a pantalla completa puede establecerse desde cualquier ventana.
- La música puede activarse o desactivarse y la opción surtirá efecto en todo el juego.
- Los efectos de sonido pueden activarse o desactivarse y la opción surtirá efecto en todo el juego.

Pruebas de navegación:

- La aplicación puede cerrarse desde cualquier ventana.
- Se puede elegir cualquiera de las opciones del menú principal.
- Es posible volver a la ventana anterior desde cualquier ventana, exceptuando la de juego.
- Es posible volver al menú principal desde la ventana de juego.
- Se puede pausar la partida y reanudarla.

- En el menú de elección de jugador no se puede continuar si no ha se ha insertado ningún jugador.
- En el menú de elección de jugador se puede cambiar el color y el tipo del jugador correctamente.
- En el menú de elección de campo se puede elegir cualquier hoyo y cualquier campo de los disponibles.
- Es posible pausar la partida mientras juega el ordenador.

Pruebas de interfaz:

- Se indica correctamente el nombre del jugador.
- Se indica correctamente el nombre del campo y el número de hoyo.
- Se actualiza de forma satisfactoria la imagen del palo y sus datos cuando se realiza un cambio.
- Se actualiza adecuadamente la información relativa al par del jugador.
- Se muestran correctamente los indicadores de fuerza y efecto.
- Se muestra correctamente el cursor de dirección.
- Es posible visualizar el mapa del campo.

Pruebas de funcionamiento:

- Se controla que la bola de golf esté dentro de los límites del campo.
- Las colisiones de la bola de golf se detectan y se procesan correctamente.
- La inteligencia artificial interpreta correctamente los diferentes campos.
- Se cumplen las restricciones de los diferentes modos de juego.

Capítulo 9

Conclusiones

A pesar del esfuerzo y la dedicación necesaria para llevar a cabo este proyecto, su realización ha merecido la pena.

He aprendido a enfrentarme a un proyecto más o menos complejo en el que ha sido necesario contar con conocimientos y habilidades de diversa índole.

La búsqueda de soluciones a los inconvenientes que han ido surgiendo durante el desarrollo me ha permitido adquirir nuevas competencias y asimilar nuevos conceptos.

En general, he cumplido con mi propósito inicial de realizar un videojuego usando herramientas libres y el resultado final ha sido, a mi juicio, satisfactorio.

9.1. Mejoras

El videojuego puede mejorarse en muchos aspectos. A continuación, comentaré algunas de las mejoras que pueden llevarse a cabo:

- Incorporar un editor de campos.
- Mayor número de jugadores.
- Permitir seleccionar los palos con los que se quiera jugar.
- Realización de nuevos modelos para los jugadores.
- Comportamiento de la bola más imprevisible.
- Partidas multijugador en línea.
- Nuevos diseños para los campos.
- Inteligencia artificial con mayor visión de juego.
- Mejora en el diseño de los menús.

- Permitir redefinir las teclas de juego.
- Mejorar las animaciones.
- Mejorar el efecto de desplazamiento de la cámara.
- Posibilidad de jugar con joystick.
- Animación de rótulos en pantalla.
- Incluir más modos de juego.
- Incorporar retos.
- Editor de personajes.
- Perfiles de configuración para cada usuario.
- Permitir cambiar la resolución de la pantalla.
- Portabilidad a otros sistemas.

9.2. Herramientas utilizadas

SDLgolf[2] ha sido desarrollado en C++[11][4], empleando el conjunto de bibliotecas SDL[5][8] y el analizador TinyXML[6].

Toda la edición de código se ha llevado a cabo en Gedit[3], la documentación se ha generado con Doxygen[10] y la memoria ha sido desarrollada en LaTeX[13].

Los gráficos han sido modelados en Gimp[3] y el audio se ha editado en Audacity[1].

Para la generación automática de código se ha utilizado Make[12].

El proyecto ha sido hospedado en RedIRIS[1][7], empleando el sistema de control de versiones Subversion[9].

Realizado en un sistema Debian GNU/Linux.

```
andres0x@akatosh:~$ cat /etc/debian_version
wheezy/sid
```

[1]Puede acceder desde https://forja.rediris.es/projects/sdlgolf/

Bibliografía

[1] Audacity Wiki Tutorials. http://wiki.audacityteam.org/index.php?title=Tutorials.

[2] Golf (From Wikipedia, the free encyclopedia). http://en.wikipedia.org/wiki/Golf.

[3] Guía de ayuda de Gedit. http://library.gnome.org/users/gedit/stable/.

[4] Reference of the C++ Language Library. http://www.cplusplus.com/reference/.

[5] SDL API reference. http://www.libsdl.org/cgi/docwiki.cgi/SDL_API.

[6] TinyXml Documentation. http://www.grinninglizard.com/tinyxmldocs/index.html.

[7] Wiki de Ayuda de la Forja de RedIRIS. https://forja.rediris.es/plugins/wiki/index.php?id=206&type=g.

[8] Antonio García Alba. Tutorial de libSDL para la programación de videojuegos. http://softwarelibre.uca.es/wikijuegos/.

[9] Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato. Control de versiones con Subversion. http://svnbook.red-bean.com/nightly/es/index.html.

[10] Dimitri Van Heesch. Pagina oficial de Doxygen. http://www.doxygen.org.

[11] Francisco Palomo Lozano. Inmaculada Medina Bulo. Gerardo Aburruzaga García. *Fundamentos de C++*. Servicio de Publicaciones de la Universidad de Cádiz, 2006.

[12] Gerardo Aburruzaga García. Make. Un programa para controlar la recompilación. http://www.uca.es/softwarelibre/publicaciones/make.pdf.

[13] Wikibooks. The Book of LaTeX. http://en.wikibooks.org/wiki/LaTeX.

Apéndice A

Manual de usuario

A.1. Requisitos mínimos

SDLgolf está disponible para sistemas GNU/Linux y requiere para su funcionamiento los siguientes elementos:

- Procesador de 800 Mhz.
- 256 MB de memoria principal.
- 100 MB de espacio disponible en el disco duro.
- Tarjeta gráfica (aceleración 3D no necesaria).
- Tarjeta de sonido.
- Teclado.

A.2. Instalación

Para poder llevar a cabo la instalación de SDLgolf, es necesario contar con las siguientes bibliotecas en el sistema:

- SDL 1.2: http://www.libsdl.org/download-1.2.php
- SDL_image 1.2: http://www.libsdl.org/projects/SDL_image/
- SDL_mixer 1.2: http://www.libsdl.org/projects/SDL_mixer/
- SDL_ttf 2.0: http://www.libsdl.org/projects/SDL_ttf/
- TinyXML: http://sourceforge.net/projects/tinyxml/

La compilación se llevará a cabo mediante el compilador G++, incluido en el paquete GCC y usando la herramienta de generación de código Make:

- GCC: http://gcc.gnu.org/
- Make: http://www.gnu.org/software/make/

Si se desea generar la documentación del juego, será necesario instalar la aplicación Doxygen:

- Doxygen: http://www.stack.nl/~dimitri/doxygen/index.html

El código fuente de SDLgolf puede descargarse en la página oficial del proyecto[1], desde la sección de descargas o empleando el sistema de control de versiones Subversion:

- Subversion: http://subversion.apache.org/

Para descargar la última versión disponible en el repositorio, simplemente será necesario introducir la siguiente línea en consola:
`svn checkout https://forja.rediris.es/svn/sdlgolf`

Se creará la carpeta `sdlgolf`, desde la que podrá acceder al código fuente de el videojuego y llevar a cabo la compilación del mismo ejecutando el comando `make`. La generación de la documentación podrá realizarse ejecutando `make doc`.

A.3. Menú principal

Desde el menú principal podrá acceder a los diferentes modos de juego de SDLgolf, así como a las opciones del videojuego o a la información relativa al autor.

[1] Accesible desde: https://forja.rediris.es/projects/sdlgolf/

Figura A.1: Manual de usuario: Menú principal

Use las teclas cursor arriba o cursor abajo para seleccionar la opción que desee y pulse la tecla enter para confirmar su selección.

A.4. Modos de juego

SDLgolf cuenta con tres modos de juego, soportando hasta cuatro jugadores. A continuación veremos en qué consiste cada uno de ellos.

A.4.1. Practicar

En el modo de juego Practicar, uno o varios jugadores pueden practicar un determinado hoyo de un campo. No existen ganadores en este modo de juego.

A.4.2. Matchplay

En el modo Matchplay, dos o más jugadores se enfrentan (por parejas en caso de haber más de dos jugadores) en todos los hoyos de un campo. El equipo o el jugador que gane un hoyo sumará un punto. El equipo o el jugador que pierda el hoyo no sumará nada. Si los equipos o los jugadores empatan un hoyo, tampoco sumarán puntos. Los miembros de un mismo equipo podrán compartir la bola. Si se comparte la bola, los jugadores también compartirán el par. Ganará la partida el equipo o el jugador que más puntos haya conseguido.

A.4.3. Strokeplay

En el modo de juego Strokeplay, uno o más jugadores se enfrentan en todos los hoyos de un campo. Al final de cada hoyo, cada jugador obtendrá una puntuación determinada por la resta del par del hoyo menos el par del propio jugador. Ganará la partida el jugador que obtenga la puntuación más baja, es decir, el jugador que haya completado todos los hoyos de un campo haciendo menor par que sus contrincantes.

A.5. Elección de jugador

Desde el menú de elección de jugador podrá establecer el nombre, color y tipo de el jugador o los jugadores que participarán en la partida.

Para agregar un nuevo jugador a la partida, pulse la tecla enter en cualquiera de los recuadros de nombre disponibles.

El sistema comenzará a capturar el texto que introduzca desde el teclado. Cuando haya terminado de introducir el nombre del nuevo jugador, vuelva a pulsar la tecla enter para confirmarlo.

Figura A.2: Manual de usuario: Captura de texto desde el teclado

Se le asignará un color aleatorio y el tipo Humano. Si desea cambiar el color o el tipo de jugador que acaba de añadir, seleccione el recuadro pertinente y vaya pulsando la tecla enter hasta encontrar la opción deseada.

Si desea editar el nombre de un jugador, seleccione su nombre y pulse la tecla enter. Una vez finalizada la edición, vuelva a pulsar la tecla enter para confirmarla.

Si desea crear un jugador de forma aleatoria, seleccione un recuadro de nombre y presione la tecla espacio. El sistema creará un jugador con un nombre, tipo y color aleatorio.

Figura A.3: Manual de usuario: Jugadores creados de forma aleatoria

Para eliminar a un jugador, seleccione su nombre y pulse la tecla borrar. También puede eliminar a un jugador editando su nombre e insertando uno vacío.

Si ha seleccionado el modo de juego Matchplay y en la partida participan más de dos jugadores, deberá seleccionar la pareja del primer jugador.

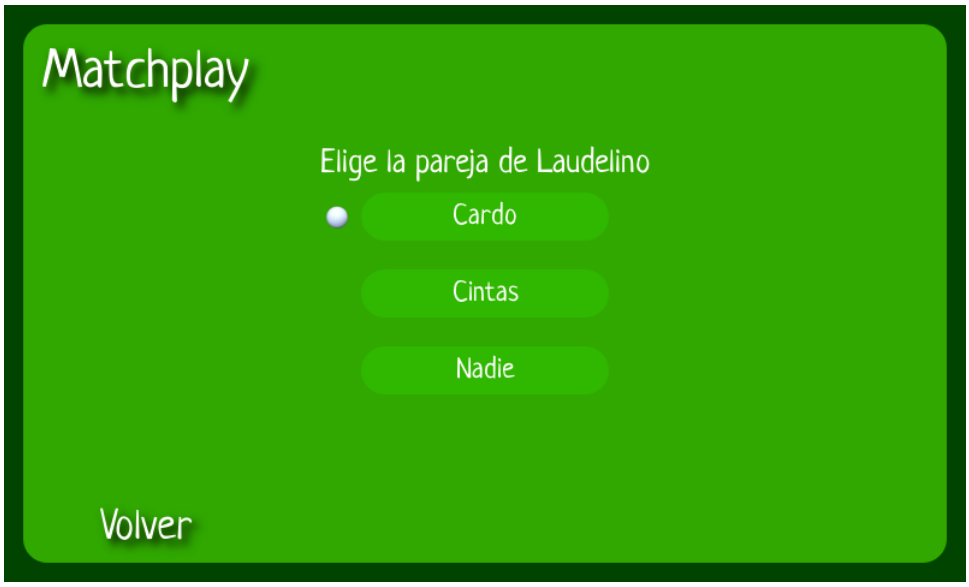


Figura A.4: Manual de usuario: Elección de pareja en el modo de juego Matchplay

Una vez elegida la pareja, podrá seleccionar el tipo de Matchplay que desea jugar.

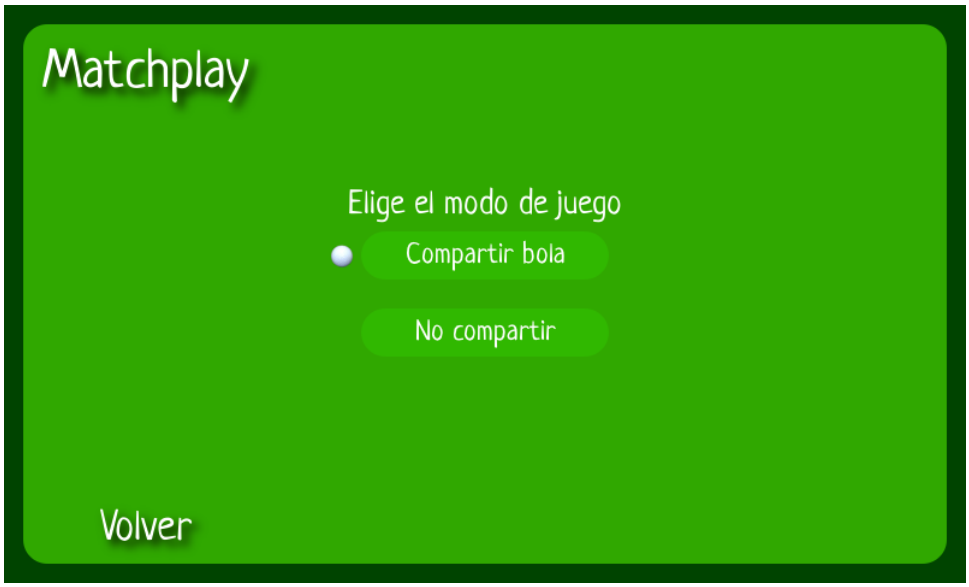


Figura A.5: Manual de usuario: Elección de tipo de Matchplay

A.6. Elección de campo

Desde el menú de elección de campo podrá seleccionar el hoyo y el campo en el que desea jugar. Use las teclas cursor arriba y cursor abajo para seleccionar el nombre del campo o el número de hoyo. Una vez seleccionado, pulse la tecla enter para cambiar su elección. Cuando esté preparado para comenzar la partida, seleccione jugar y pulse la tecla enter.

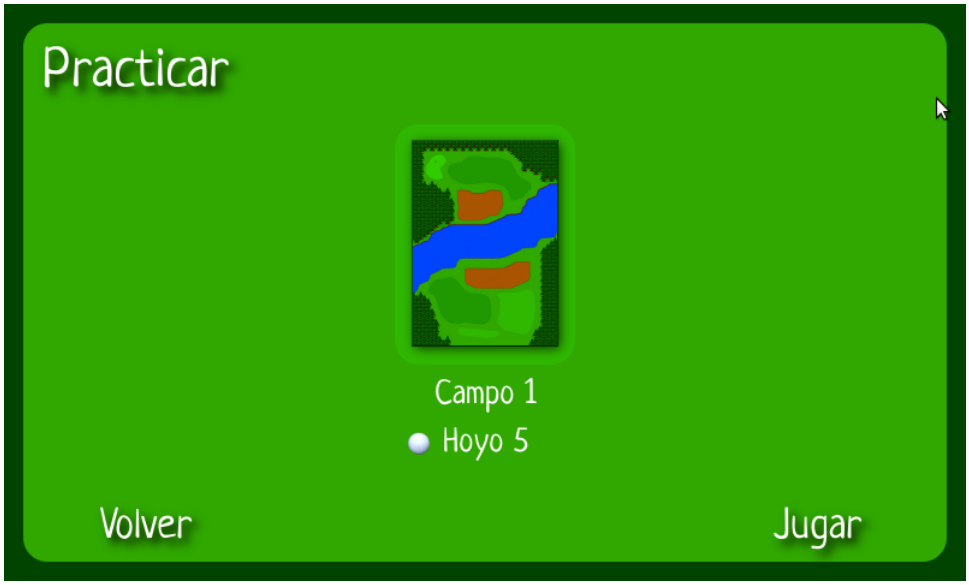


Figura A.6: Manual de usuario: Selección de otro hoyo del campo

A.7. Opciones

Desde el menú de opciones podrá activar o desactivar los efectos de sonido y la música. Para ello, seleccione la opción deseada y pulse la tecla enter para activar o desactivar dicha opción.

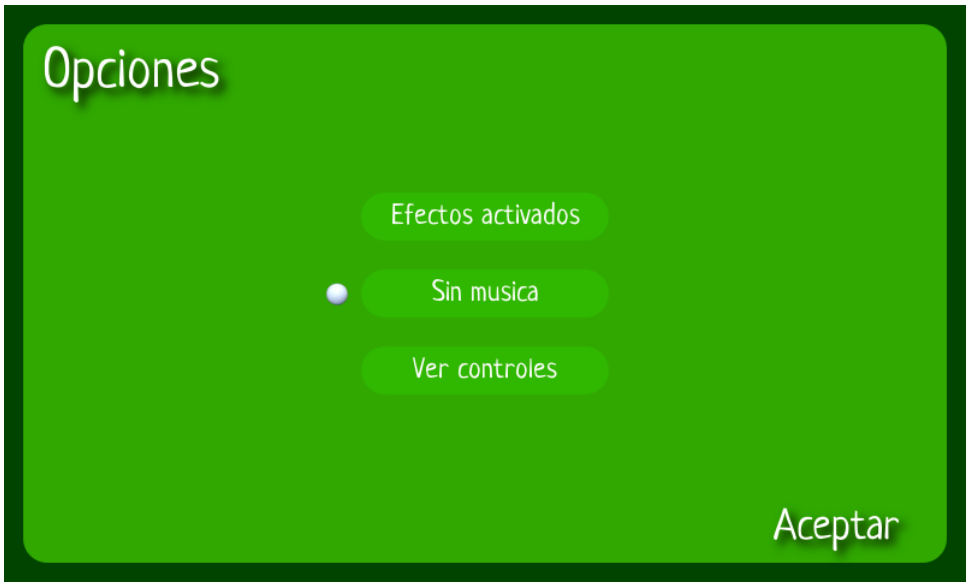


Figura A.7: Manual de usuario: Desactivación de la música del juego

Seleccione aceptar y pulse enter cuando esté listo. Las opciones seleccionadas se guardarán de forma automática.

Si desea, también podrá consultar los controles del juego.

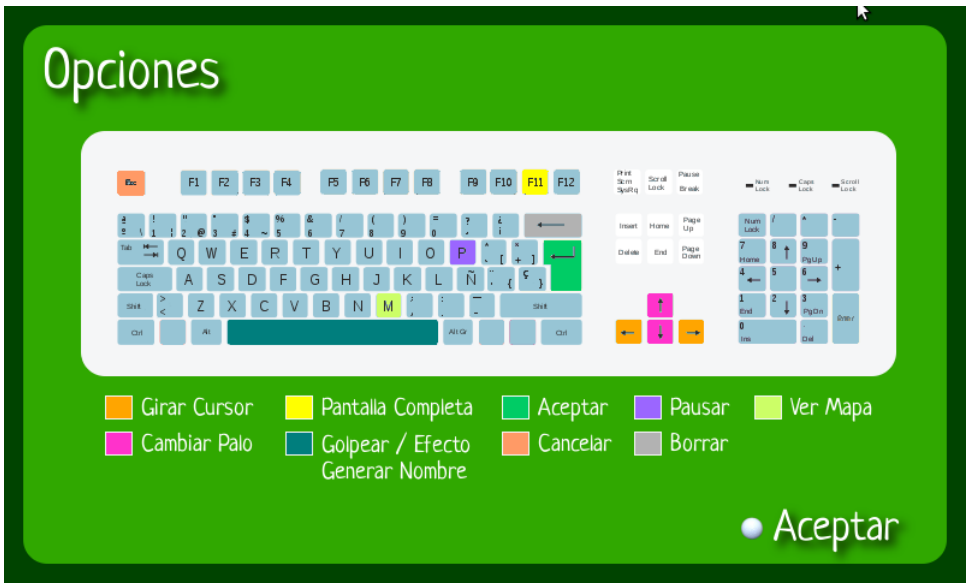


Figura A.8: Manual de usuario: Visualización de los controles del juego

A.8. Acerca de

Este menú le permitirá acceder a información relativa a SDLgolf. Pulse la tecla enter cuando desee volver al menú principal.

Figura A.9: Manual de usuario: Información sobre el juego

A.9. Controles

Cuando inicie la partida se le mostrará la ventana de juego. Desde ella podrá visualizar toda la información relativa al jugador y a la partida actual.

Figura A.10: Manual de usuario: Partida de práctica

A.9.1. Cambiar la dirección del cursor

Si desea cambiar la dirección a la que apunta el jugador, utilice las teclas cursor izquierda para girar el cursor a la izquierda y cursor derecha para girar el cursor a la derecha.

A.9.2. Cambiar el palo de golf

El palo de golf puede cambiarse empleando las teclas cursor arriba y cursor abajo. Con cursor arriba podrá seleccionar el palo siguiente al actual y con cursor abajo el palo anterior.

A.9.3. Realizar lanzamiento

Si desea llevar a cabo el lanzamiento de la bola de golf, pulse la tecla espacio. Se le mostrará el indicador de nivel de fuerza. Utilice la tecla espacio para determinar el nivel de fuerza que desea aplicar al lanzamiento.

Figura A.11: Manual de usuario: Indicador de nivel de fuerza

Una vez seleccionado el nivel de fuerza, se le mostrará el indicador de efecto. Utilice la tecla espacio para determinar el sentido del efecto que desea aplicar al lanzamiento.

Figura A.12: Manual de usuario: Indicador de efecto

El jugador llevará a cabo el lanzamiento atendiendo a los niveles de fuerza y efecto seleccionados.

A.9.4. Visualizar el mapa

Para visualizar el mapa del hoyo en el que está jugando, pulse la tecla m. En el mapa aparecerá la posición del jugador activo y su cursor, así como la localización del hoyo del campo.

Figura A.13: Manual de usuario: Visualización del mapa del hoyo

A.9.5. Pausar el juego

Puede pausar el juego pulsando la tecla escape o la tecla p. Seleccione continuar y pulse enter cuando esté listo. Si desea volver al menú principal, seleccione la opción salir y pulse enter. SDLgolf pedirá que confirme la salida.

Figura A.14: Manual de usuario: Partida en pausa

A.9.6. Establecer el modo a pantalla completa

Si desea que SDLgolf se ejecute a pantalla completa, pulse la tecla F11 en cualquier ventana.

Apéndice B

Formato de los documentos XML

Con la intención de hacer SDLgolf fácilmente ampliable y extensible, se emplean documentos XML para el almacenamiento de datos.

A continuación, veremos la estructura interna de estos archivos y una breve descripción de los mismos.

B.1. Campos

El documento `campos.cfg` incluye todos los datos relacionados con todos los campos disponibles en el juego. Su estructura es muy intuitiva.

```
<?xml version="1.0" ?>
<Campos>
 <Campo nombre="Campo1" ruta="ruta1" colores="colores.xml" animaciones="estandar">
  <Hoyos>
   <Hoyo numero="1" par="4" x_bola="10" y_bola="10" x_hoyo="100" y_hoyo="100" />
   <Hoyo numero="2" par="4" x_bola="10" y_bola="10" x_hoyo="100" y_hoyo="100" />
  </Hoyos>
 </Campo>
 <Campo nombre="Campo2" ruta="ruta2" colores="colores.xml" animaciones="estandar">
  <Hoyos>
   <Hoyo numero="1" par="4" x_bola="10" y_bola="10" x_hoyo="100" y_hoyo="100" />
  </Hoyos>
 </Campo>
</Campos>
```

En la entidad `Campo`, encontramos los siguientes atributos:

- **nombre**: Nombre que tendrá el campo en el juego.
- **ruta**: Ruta a la carpeta de las imágenes del campo (dentro del directorio de imágenes).
- **colores**: Ruta al fichero de colores de colisión (dentro del directorio de configuración).
- **animaciones**: Ruta al directorio de las animaciones del campo (dentro del directorio de animaciones).

En la entidad `Hoyo`, se distinguen los siguientes atributos:

- **numero**: Número de hoyo.
- **par**: Par del hoyo.
- **x_bola**: Coordenada *x* de la posición de salida de los jugadores.
- **y_bola**: Coordenada *y* de la posición de salida de los jugadores.
- **x_hoyo**: Coordenada *x* de la posición del hoyo del campo.
- **y_hoyo**: Coordenada *y* de la posición del hoyo del campo.

B.2. Colores

El documento `colores.cfg` incluye todos los colores de colisión para todos los elementos del campo de golf.

```
<?xml version="1.0" ?>
<Colores>
 <Elemento nombre="agua">
  <color r="0" g="0" b="1"/>
  <color r="0" g="1" b="0"/>
 </Elemento>
 <Elemento nombre="arboles">
  <color r="0" g="1" b="1"/>
 </Elemento>
 <Elemento nombre="bunker">
  <color r="1" g="0" b="0"/>
 </Elemento>
 <Elemento nombre="hoyo">
  <color r="1" g="0" b="1"/>
 </Elemento>
</Colores>
```

La entidad `Elemento` contiene un único atributo que es el nombre del elemento. Para cada `color` del elemento se guardan los valores RGB del mismo.

B.3. Opciones

En el fichero `opciones.cfg` se guarda el estado de los efectos de sonido y la música del juego.

```
<?xml version="1.0" ?>
<Opciones>
    <sonido efectos="si" musica="si" />
</Opciones>
```

GNU Free Documentation License

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed

as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page"

means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you

may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the

Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with . . . Texts.” line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.